

DOCUMENT RESUME

ED 238 430

IR 050 544

AUTHOR Aagaard, James S.; And Others
TITLE Application Level Protocol Development for Library and Information Science Applications. Volume 1: Service Definition. Volume 2: Protocol Specification. Report No. TG.1.5; TG.50.
INSTITUTION Council on Library Resources, Inc., Washington, D.C.
PUB DATE 29 Oct 82
GRANT CLR-2033
NOTE 200p.
PUB TYPE Legal/Legislative/Regulatory Materials (090) -- Reports - Descriptive (141)
EDRS PRICE MF01/PC08 Plus Postage.
DESCRIPTORS Bibliographies; *Computer Oriented Programs; Computer Programs; *Design Requirements; *Information Networks; Information Retrieval; Information Storage; *Library Services; Man Machine Systems; *Models; *Online Systems; Recordkeeping
IDENTIFIERS Electronic Mail; Linked Systems Project; *Open Systems Interconnection

ABSTRACT

This two-volume document specifies a protocol that was developed using the Reference Model for Open Systems Interconnection (OSI), which provides a framework for communications within a heterogeneous network environment. The protocol implements the features necessary for bibliographic searching, record maintenance, and mail transfer between end-user application processes in library and information sciences. The specifications result from a grant to develop application and presentation level protocols to support the Linked Systems Project that was established among the Washington Library Network, the Research Libraries Group, and the Library of Congress. Volume 1 provides an introduction, a description of the OSI Reference Model, and an overview of the model application layer. Facilities of the application service are defined, including those for connection establishment and termination, information retrieval, record maintenance, mail, interrupt, and settings. Service descriptions are provided and the service specification technique explained. Service element specifications, interface event sequences, and parameters for service elements are outlined. Volume 2 provides an introduction and overview of the protocol, explains the protocol specification method and notation, and details the formal specification of the protocol and 33 protocol data unit (PDU) formats. A 59-item reference list is included in both volumes.
(LMM)

* Reproductions supplied by EDRS are the best that can be made *
* from the original document. *

ED238430

U.S. DEPARTMENT OF EDUCATION
NATIONAL INSTITUTE OF EDUCATION
EDUCATIONAL RESOURCES INFORMATION
CENTER (ERIC)

• This document has been reproduced as
received from the person or organization
originating it.

Minor changes have been made to improve
reproduction quality.

• Points of view or opinions stated in this docu-
ment do not necessarily represent official NIE
position or policy.

Report no. TG.1.5

Application Level Protocol Development
for
Library and Information Science Applications
Volume 1
Service Definition

Council on Library Resources

Grant 2033

Prepared by:
James S. Aagaard
Kenton E. Andersen
Wayne E. Davison

October 29, 1982

PERMISSION TO REPRODUCE THIS
MATERIAL HAS BEEN GRANTED BY

Council on Library
Resources

TO THE EDUCATIONAL RESOURCES
INFORMATION CENTER (ERIC)

IR050544

TABLE OF CONTENTS

1.	INTRODUCTION	1
2.	OSI REFERENCE MODEL	3
2.1	Layering Principles	3
3.	OVERVIEW OF THE APPLICATION LAYER	6
3.1	Role of the Application Layer in the OSI Model	6
3.2	General Characteristics of Application Service	6
3.3	Subapplications	7
4.	FACILITIES OF THE APPLICATION SERVICE	9
4.1	Connection Establishment Facility	9
4.2	Connection Termination Facility	9
4.3	Information Retrieval Facility	9
4.4	Record maintenance Facility	9
4.5	Mail Facility	10
4.6	Interrupt Facility	10
4.7	Settings Facility	10
5.	SERVICE DESCRIPTIONS	11
5.1	Connection Services	11
5.2	Information Retrieval Services	11
5.3	Record Maintenance Services	13
5.4	Mail Services	14
5.5	General Services	15
6.	SERVICE SPECIFICATION TECHNIQUE	16
6.1	Model	16
6.2	Definitions	16

6.3	Service Specification Method	17
7.	SPECIFICATION OF THE SERVICE ELEMENTS	18
7.1	Connection Establishment Facility	18
7.1.1	A_ESTABLISH Service Element	18
7.2	Connection Termination Facility	18
7.2.1	A_ABORT Service Element	18
7.2.2	A_RELEASE Service Element	18
7.3	Information Retrieval Facility	19
7.3.1	A_SEARCH Service Element	19
7.3.2	A_PRESENT_RECORDS Service Element	19
7.3.3	A_DISCARD_RESULT Service Element	19
7.3.4	A_FETCH_RECORD Service Element	20
7.4	Record Maintenance Facility	20
7.4.1	A_ADD_RECORD Service Element	20
7.4.2	A_REPLACE_RECORD Service Element	21
7.4.3	A_DELETE_RECORD Service Element	21
7.4.4	A_ADD_VALIDATED Service Element	21
7.4.5	A_REPLACE_VALIDATED Service Element	22
7.4.6	A_DELETE_VALIDATED Service Element	22
7.5	Mail Facility	22
7.5.1	A_SEND_MAIL Service Element	22
7.5.2	A_MAIL_READ Service Element	23
7.6	Interrupt Facility	23
7.6.1	A_CANCEL Service Element	23
7.7	Settings Facility	23
7.7.1	A_SET Service Element	23

7.7.2	A_SHOW Service Element	24
8.	PARAMETERS FOR SERVICE ELEMENTS	25
8.1	General	25
8.2	Formal Description of the Search Query	30
9.	SPECIFICATION OF INTERFACE EVENT SEQUENCES	33
9.1	Connection	33
9.2	Information Retrieval	35
9.3	Record Maintenance	37
9.4	Mail	41
	REFERENCES	43

1. INTRODUCTION

In order to achieve long-range goals of linking the central computing capabilities of bibliographic networks to each other and to local computing systems at libraries which are members of these networks, it is necessary to establish communication protocols which will be accepted and used by all of the participants.

As a first step in this process the Telecommunications Committee of the American Library Association began work in 1975 on a protocol for computer-to-computer communications. In 1976 a task force was established by the National Commission on Libraries and Information Science (NCLIS), in cooperation with the National Bureau of Standards (NBS), to continue the work on communication protocols. Although the protocol proposed by the NCLIS/NBS group [NTAG-77-80]¹ was described as an application protocol, in reality it includes features of what became defined as several levels in the layered ANSI/ISO Reference Model for Open Systems Interconnection which was taking form at the same time (see section 2).

An organization which was then called the Network Advisory Group (to the Library of Congress) established a task force of technical staff from library networking organizations called the Network Technical Advisory Group (NTAG). This group, which began meeting in 1977, examined possible mechanisms by which the goal of interconnection might be achieved.

In the fall of 1977 the Library of Congress awarded a contract to the New York Public Library (NYPL) to begin some of the work recommended by the NCLIS/NBS and NTAG groups. NYPL subcontracted with experts in the field of library automation; this group became known as the Protocol Implementation Planners, and produced a final report in August 1978 [NTAG-78-21]. There appears to be no recorded attempt to further refine or implement the recommendations of these three groups. Further work has been undertaken by the American National Standards Institute, Committee Z39, Subcommittee D (ANSI Z39D) and by the National Bureau of Standards Institute for Computer Sciences and Technology (NBS/ICST).

The establishment of the Linked Systems Project (LSP) among the Washington Library Network, the Research Libraries Group, and the Library of Congress, has generated a new impetus for a resolution of the protocol problem. In July 1981 the Council on

¹The notation [...] is used in this document to reference an item in the bibliography.

Library Resources awarded grant 2033 to the Northwestern University Library to develop application and presentation level protocols which support the needs of Linked Systems Project, and upon which other library and information science applications may be built. The statement of these requirements may be found in [LASP-39].

Some of the requirements for this grant are:

- 1) A set of protocols will be developed to permit the exchange of messages between two applications running on different computer systems. Initially, the protocols must support Search and Retrieval, Record Maintenance, and Mail Transfer. The protocols must be extensible to File Distribution, Interlibrary Loan, and Book Ordering.
- 2) The protocols developed must follow the principles of Open Systems Interconnection as specified in the ANSI/ISO Reference Model for Open Systems Interconnection.
- 3) The protocols developed must reside in the application and presentation layers of the OSI Reference Model.
- 4) The development will be coordinated with ANSI Z39 Subcommittee D (Computer-to-Computer Protocols).
- 5) The development should draw from the protocol development work done by ISO, ANSI, and NBS to the fullest extent possible.

2. OSI REFERENCE MODEL

This protocol was developed using the architecture formulated in the Reference Model for Open Systems Interconnection (OSI) [DP7498]. The architecture was developed by the International Organization for Standardization to provide a framework for communications within a heterogenous network environment.

2.1 Layering Principles

The model calls for a hierarchy of several layers at each node in the network for communication processing. Figure 1 shows a schematic for the layering.

Layering is a technique which imposes an order and structure on the functions required to support computer-to-computer communication. All of the functions must be performed; layering clarifies the problem and simplifies the interaction between these functions.

Some of the benefits of layering are:

- the ability to "divide and conquer" a complex technical problem
- reduction of complexity by imposing a discipline on the interactions between system components by defining and limiting those interactions
- definition of a standard nomenclature for discussion and design of network systems
- definition of a standard architecture that facilitates and encourages the development of standard procedures for all levels
- the ability to replace any single building block with a better one without disturbing any of the others
- simplified error detection and program debugging

In deciding on the number and definition of the layers in the OSI architecture, the designers followed these guidelines:

- 1) Create only as many layers as necessary.
- 2) Create separate layers to handle functions that are significantly different in the processes they perform or the technology they use.

- 3) Put functions that are substantially similar in the same layer.
- 4) Create a separate layer where there is a need for a different level of abstraction in handling the data.
- 5) Create a layer of related functions in such a way that the layer could be totally redesigned to take advantage of advances in technology without changing any of the other layers.
- 6) Define the boundary between layers at a point that minimizes the number of interactions that must take place across that boundary.
- 7) Define boundaries at the points where they have occurred and been useful in the past.
- 8) Define a boundary where it may be useful to standardize the interaction with the adjoining layer.
- 9) Design boundaries so that each layer need only interact with its two adjoining layers, one above and one below.

A layer only provides services to the layer immediately above it. The point at which two layers meet is termed the "interface". A given layer provides services to the layer above through the upper interface, and requests the services of the lower layer through the lower interface.

Each system engaged in a connection will have the seven layers defined in the Reference Model. An entity is a process or module that performs the services required of the layer within a single system. Entities in the same layer are called "peer entities".

Peer entities communicate according to a set of rules. This set of rules is termed the "protocol" of the layer. The protocol provides for a predefined set of procedures and formats. Through this protocol, peer entities can control the flow of information. A unit of data, in a predefined format, passed between two peer entities is called a "protocol data unit", or "PDU".

Control information and data will be passed to the peer entities using the services of the lower layers. The lower layers provide a virtual link between the peer entities. Communication actually proceeds down through the lower layers, across the physical link, and up through the layers until it reaches the peer entity of the associated layer.

To accomplish this, each layer (except the application layer) provides a connection for the delivery of the PDUs of the next higher layer. It does this by inserting the higher layer's PDU into one or more of its own PDUs, and then requesting the delivery of its PDUs by the next lower layer¹. Each peer entity, upon receiving a PDU, extracts and assembles the higher layer's PDU and passes it to the higher layer.

Level. Layer

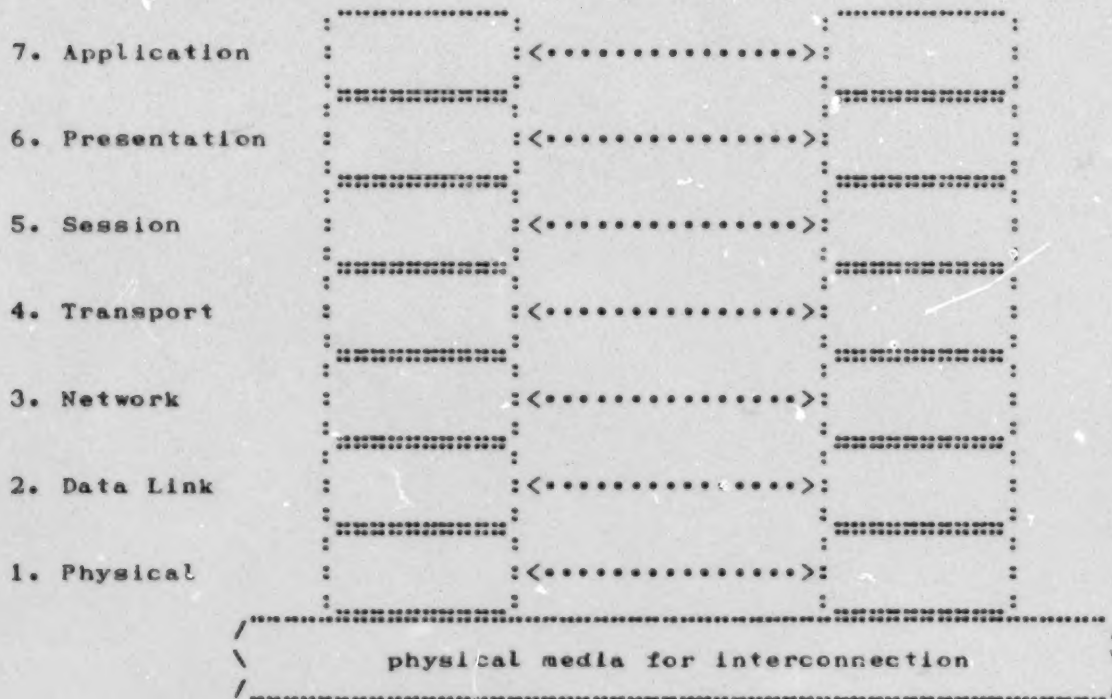


Figure 1. OSI Model

¹note that the physical layer actually transfers the data across the physical medium

3. OVERVIEW OF THE APPLICATION LAYER

3.1 Role of the Application Layer in the OSI Model

The protocol discussed in this document resides at the seventh level of the OSI model -- the application layer. It implements the features necessary for bibliographic searching, record maintenance, and mail transfer between end-user application processes in library and information sciences.

As the highest layer in the OSI Model, the application layer provides services not to a next higher layer, but to the end-user application processes which are the users of the OSI environment. Protocols at this layer are generally industry specific.

All user specifiable parameters of each communications connection are made known to the OSI environment via the application layer. All specification of the semantics of information exchanged not otherwise implicit among users of the OSI environment is provided via the application layer. Conversely, as the only layer in the OSI environment that directly provides services to the end-user application processes, the application layer necessarily provides all services that are directly accessible by the end-user application processes. These services include, among others:

- a) identification of intended communications partners
- b) determination of current availability of an intended partner
- c) establishment of authority to communicate
- d) agreement on privacy mechanisms as required
- e) authentication of intended partners
- f) determination of cost allocation methodology
- g) determination of adequacy of required resources
- h) determination of acceptable quality of service
- i) synchronization of cooperating partners
- j) selection of dialog discipline, including initiation and release procedures
- k) agreement on responsibility for error recovery
- l) agreement on procedure for data validity commitment
- m) identification of constraints on data syntax
- n) information transfer

3.2 General Characteristics of Application Service

The service is designed to handle program-to-program communications. Program-to-terminal and terminal-to-terminal communications are not addressed by the service.

The service supports online interaction.

The service does not provide explicitly for bulk transfer of data. Although bulk transfer could be done using the proposed record maintenance features, it is likely that more efficient methods could be designed using emerging file transfer services.

The communications path between two programs is called a connection. Each connection must be established explicitly. Connections are normally terminated explicitly, but are also terminated by the failure of a node or of the communications link.

The service provides for only point-to-point connections. Each connection consists of an originating application and a target application. These roles may not be reversed within a connection.

A connection cannot be restarted, i.e., no status information (e.g. parameter settings) is retained between connections. A new, logically different connection must be established.

Each connection is single-threaded, i.e., a program may make only one request at a time. A program may continue to make requests without establishing a new connection.

The service allows an application to establish multiple simultaneous connections with another application. At the same time the application can have connections with other applications.

The service supports the transmission of 8-bit binary, bit string, and character string data. As character string data, all implementors must support the eight-bit library extension to the American Standard Code for Information Interchange (commonly known as ALA-MARC), as described in Appendix III.B.1 of [MFED].

3.3 Subapplications

The Linked Systems Project requires intersystem communication in 5 distinct subapplications, supporting 3 facilities:

<u>Facility</u>	<u>Subapplication</u>
Information retrieval	Information retrieval
Record maintenance	Requests Reporting

Mail

Transmission
Read notification

Requiring the originating application to select a subapplication simplifies the design of the target application, since the target application will not be expected to handle a mail transmission request in between two information retrieval requests, for example. The specification of subapplications also simplifies the addition of new features to the protocol.

Switching between subapplications is not supported. Only one subapplication may be active within a connection.

The five subapplications plus connection establishment and connection termination constitute seven distinct phases in which the service operates. Some service elements are general; they may be used in more than one phase. Others may be used only in the appropriate phase.

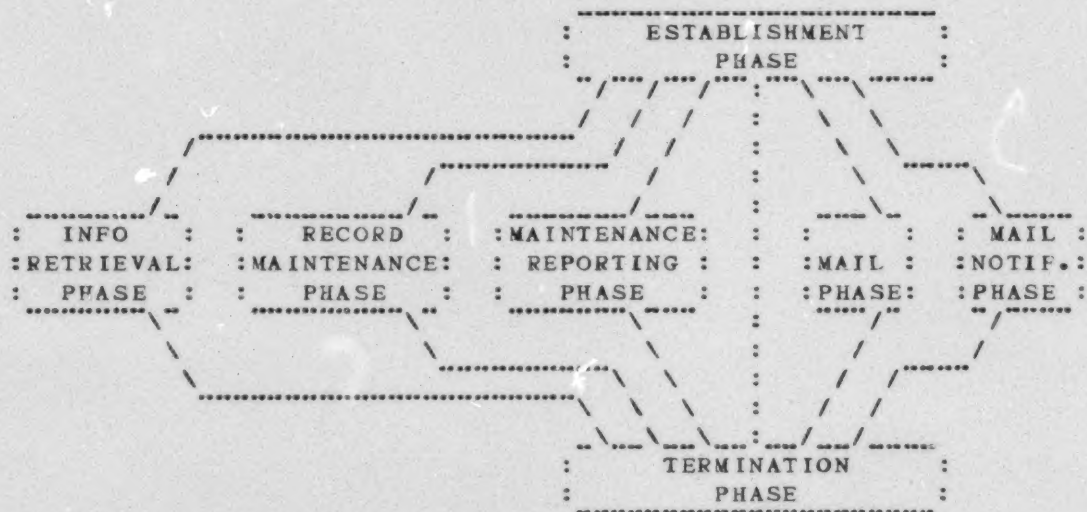


Figure 2. Service phases

4. FACILITIES OF THE APPLICATION SERVICE

4.1 Connection Establishment Facility

The connection establishment facility enables an end-user application process to establish an application connection with another end-user application process. During the establishment of this connection, the end-user processes can exchange values of application connection parameters. These parameters include the subapplication to be used during this connection and parameters associated with that subapplication; e.g. file name, limits on the size of retrieval sets, immediate or deferred processing, etc.

4.2 Connection Termination Facility

The connection termination facility provides for orderly termination by mutual agreement of the two end-user processes, and for unilateral termination initiated by one of the end-user processes, or upon detection of an exception condition by one of the application entities or an entity in a lower layer. Unilateral termination may be disruptive and result in the loss of data in transit. During the termination of a connection, the end-user processes may exchange statistics about activity which occurred during the life of the connection.

4.3 Information Retrieval Facility

The information retrieval facility enables an end-user process to make search requests against databases held by another end-user process, to receive information about the results of these requests, modify the results by means of further search requests, to store these results for later use, and to transfer groups of records according to their place in a set of results or according to their key.

4.4 Record maintenance Facility

The record maintenance facility enables an end-user process to transmit records for addition to the database of another end-user process, or for alteration or deletion of records already in that database. Further, this facility provides for reporting the results of immediate or deferred database update processing of the transmitted records back to the originating process.

4.5 Mail Facility

The mail facility enables an end-user process to transmit electronic mail messages to another end-user process, and provides for reporting back to the originator the reading of such messages.

4.6 Interrupt Facility

This facility enables either end-user process to interrupt the peer's processing and cancel whatever action is currently in progress, without terminating the connection. This facility can be used in conjunction with the Information Retrieval, File Maintenance, and Mail facilities.

4.7 Settings Facility

The settings facility enables an end user process to set, change, and read the parameters held by its peer. This facility can be used in conjunction with the Information Retrieval, File Maintenance, and Mail facilities.

5. SERVICE DESCRIPTIONS

5.1 Connection Services

Before two application processes may communicate, they must establish a connection. Once established, a connection may be used until terminated.

The originating end-user application process issues the A_ESTABLISH request. The originator may bid various parameters, such as the subset of application services to be used. The target end-user process must reply with either an A_ESTABLISH response or an A_ABORT request. If the target accepts connection, it may also accept the originator's parameters, or it may bid a more restrictive set, which will then be in effect. If the target aborts the connection, it will supply a reason code.

Connections may be terminated gracefully or abruptly. To terminate gracefully, either end-user process may issue the A_RELEASE request. The other end-user must reply with the A_RELEASE response. To terminate abruptly, either end-user process may issue the A_ABORT request. The application receiving an A_ABORT indication does not respond.

5.2 Information Retrieval Services

These service elements support a general query structure which can be applied to specific data files and processing capabilities by agreements between communicating applications. This structure is a mechanism for communicating search queries between applications in a mutually understandable form. It is the responsibility of the end-user application process originating a query to translate its local user query into the network form. It is the responsibility of the end-user process receiving a query to translate the network form into its internal processing form.

The result of a search query is a "result set", which is maintained by the target application. The target assigns and returns a "result set identifier" with each result set. Each target application must retain at least the most recent result set. It may also retain additional, previous result sets during the session if specified by the result set disposition subapplication parameter. If the target is retaining multiple result sets, the originating application may make an A_DISCARD_RESULT request.

The basic query concept is "Retrieve from the specified set of items those items with the properties indicated," e.g. "Find those records in the MARC Authority Format file which contain the

word Library in a corporate name," or "Find those records in MARC Books Format with the LC Catalog Card Number 76-86995."

It is assumed that certain common sets of items will be known to all host systems by a common name, e.g. MARC BOOKS. Other sets will be identified by specific names assigned by the application holding the set, e.g. Eighteenth Century Short Title Catalog. An end-user process wishing to access such a set must determine the specific name, either through a directory or through prior agreement.

Properties are composed of values with certain attributes, e.g. the value "Roots" with the attribute of being a title. The service will provide for a truncation attribute. Relational attributes may be used. Logical operators may be used to specify the relationships among various terms and/or result sets, e.g. the value "Heritage" as a title word OR the value "England" as a geographic subject. Reverse Polish Notation (RPN) is used to support nesting of logical operators, e.g. (property1 AND property2) OR property3.

RPN is a form that places logical operators after the terms on which they operate. This practice is known as "postfix operator notation". The practice provides an unambiguous statement of the order in which the operators are to be exercised, without requiring the use of parentheses. The query

"author=jones OR (author=smith AND date>1947)"

would be expressed in RPN as

"author=jones author=smith date>1947 AND OR".

RPN is usually processed by storing the intermediate results on a stack. The steps in processing the query would be:

- a) Process the first item "author=jones" and place the result set on the (empty) stack.
- b) Process the second item "author=smith" and place the result set on the stack.
- c) Process the third item "date>1947" and place the result set on the stack.
- d) Process the fourth item "AND" against the top two result sets on the stack (b and c, above), replacing them with the single result set.
- e) Process the fifth item "OR" against the top two result sets on the stack (a and d, above), replacing them with the single (final) result set.

RPN is a convenient way for an application process to construct a query, since it makes it simple to ensure the correct order of evaluation, and does not require the program to format a text string.

An end-user process using these services must recognize all of the standard parameters specifying item sets and properties but it is not required to support all of them. The service provides a "feature not supported" return code. There also is an option for the target end-user process to reply with a suggested alternate form of the search query which the target does support.

The originating end-user process is allowed to specify the manner in which the results of a search query and the retrieved items are transferred back to the origin. A "fullness code" is provided to indicate which of several levels of record completeness will be provided. Options include:

- transmission of only the count of the number of items retrieved

- specification of full or brief form of the items retrieved

- transmission of both the count and some number of items in form specified

- transmission of a specified number of retrieved items, with the ability to specify an offset from the beginning of the result set

An A FETCH RECORD request is provided which enables an originating end-user process to retrieve a single item by specifying a unique key for that item. This type of retrieval does not affect the contents of the result set for a search query in progress. It is possible to specify either the full or brief form for transmission of an item retrieved in this manner.

5.3 Record Maintenance Services

Although the service elements described in this section are oriented specifically toward the needs of the Linked Systems Project, as described in [LASP-39], it is believed that they will be adequate for other library network environments.

The record maintenance services provide the capability for the originating application to request changes in files located at the target application. The basic services which are provided are A ADD RECORD, A REPLACE RECORD, and A DELETE RECORD.

Each of the three service elements provides for parameters which specify the name of the file on the target application and the key of the record.

The originating application may ask for acknowledgement only of the receipt of the record, or acknowledgements of both the receipt and validation of the record. When both receipt and validation acknowledgements are to be made, the receipt acknowledgement will include a transaction reference number, so that the validation acknowledgement (which must be sent over a separate connection originated by the validating system) may be identified with the original request. A "negative" validation response may include a copy of the record with the fields in error marked.

A "negative" receipt return code may be used under conditions where the same request might be accepted if submitted at a later time (such as due to a temporary shortage of buffer space).

Requests which involve replacement or deletion of records also may include version numbers which will participate in the validation.

5.4 Mail Services

It is assumed that the message format described in [CBOS-80-02] will be used.

The terms "sender" and "recipient" refer to individuals, not end-user application processes. Names of mail users will be commonly known, or will be available through directories maintained by the target.

The originating end-user process collects the sender's message, determines the target identification, and issues the A SEND MAIL request. The target end-user process receives the message, verifies the recipient's identifier, delivers the message (if possible), and replies with the A SEND MAIL response.

If the message is to be sent to more than one target, the originator must make an A SEND MAIL request to each target.

A message may be addressed to one or more recipients at the same target. The target end-user process is responsible for delivering a copy of the message to each recipient.

If the message is related to a particular record or group of records, that information must be contained in the free text of the message.

A recipient of a message may edit it or append text as required. The message may then be sent through the network as if it were an original message.

5.5 General Services

Certain requests have meaning in any sub-application. The establishment and termination of connections and sub-applications have already been described.

An A CANCEL request is provided so that the originating application may request the termination of processing by the target application. The target responds with a code indicating its processing status before interruption.

6. SERVICE SPECIFICATION TECHNIQUE

6.1 Model

A layer service is defined in terms of an abstract model having the following elements:

- a) service users (i.e. application programs)
- b) the service provider (i.e. application layer)

Each service user gains access to the service provider at one or more service access points (denoted "+" in Figure 3), each of which is uniquely identified by a service access point address.

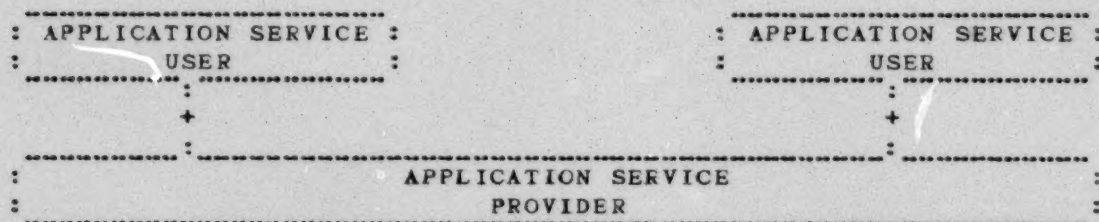


Figure 3. Service Model

All interactions are defined between the two service-users located at separate service access points. The service users communicate via the service provider in the Application layer. Information is passed between a service-user and the service provider by means of primitives.

The occurrence of a primitive is a logically instantaneous and indivisible event, which cannot be interrupted by another event.

One or more parameters may be associated with a primitive and each of these parameters has a defined range of values.

6.2 Definitions

The service specification presented in this report makes use of the following terms.

phase: A period of time during which a particular group of services are valid.

facility: A functionally-related group of one or more service elements.

service element: The smallest unit of service which can be invoked at a service access point. Each invocation of a service element results in a specific sequence of primitives at the service access points and actions within the service (including protocol exchanges). Thus, a service element corresponds to a group of functionally related primitives.

primitive: An abstract, implementation-independent element of interaction between a service user and the service provider. There are four types of primitives:

- 1) request: A primitive issued by a service user to invoke some procedure.
- 2) indication: A primitive issued by a service provider to (a) invoke some procedure, or (b) indicate that a procedure has been invoked by the service user at the peer service access point
- 3) response: A primitive issued by a service user to complete some procedure previously invoked by an indication at that service access point
- 4) confirmation: A primitive issued by a service provider to complete some procedure previously invoked at that service access point.

6.3 Service Specification Method

For each service facility, the service elements which make up that facility are described. For each service element, the phases in which it is valid are listed, and the available parameters are tabulated. The table shows to which primitive the parameters apply. An X indicates that the parameter applies to the request/indication primitives and/or the response/confirm primitives. Brackets indicate that the parameter is optional. This information appears in section 7. A description of all parameters is given in section 8. The allowable sequences of primitives are specified in section 9.

All services are confirmed except A_ABORT and A_CANCEL (the application acknowledges the cancellation by returning a response of "cancelled" to the original request). The following section shows which primitives exist for each service and which parameters are applicable to each primitive.

7. SPECIFICATION OF THE SERVICE ELEMENTS

7.1 Connection Establishment Facility

7.1.1 A_ESTABLISH Service Element

Function: Establish an application connection.

Valid phases: Establishment

Primitives and parameters:

<u>Parameter</u>	<u>Req/Ind</u>	<u>Res/Con</u>
Called_application	X	
Connection_parameters	X	[X]
Subapplication_ID	X	
Subapplication_parameters	[X]	[X]

7.2 Connection Termination Facility

7.2.1 A_ABORT Service Element

Function: Terminate a connection immediately without regard for any work in process.

Valid phases: any

Primitives and parameters:

<u>Parameter</u>	<u>Req/Ind</u>
Abort_reason	X

7.2.2 A_RELEASE Service Element

Function: Initiate a graceful termination of an application connection.

Valid phases: Information Retrieval
Record Maintenance Request and Reporting
Mail transmission and Read Notification.

Primitives and parameters:

<u>Parameter</u>	<u>Req/Ind</u>	<u>Res/Con</u>
Release_reason	X	
Statistics	[X]	[X]

7.3 Information Retrieval Facility

7.3.1 A_SEARCH Service Element

Function: Request a search of a database at the target system.

Valid phases: Information retrieval

Primitives and parameters:

<u>Parameter</u>	<u>Req/Ind</u>	<u>Res/Con</u>
Query	X	
Search_status		X
Result_count		X
Result_set_ID		[X]
Fullness_code		[X]
Search_diagnostic		[X]
Alternate_query		[X]
Number_of_records		[X]
Data		[X]

7.3.2 A_PRESENT_RECORDS Service Element

Function: Request the return of records retrieved by a search.

Valid phases: Information Retrieval

Primitives and parameters:

<u>Parameter</u>	<u>Req/Ind</u>	<u>Res/Con</u>
Fullness_code	X	
Result_set_position	[X]	[X]
Number_of_records	[X]	[X]
Result_set_ID	[X]	
Data		X
Present_status		X

7.3.3 A_DISCARD_RESULT Service Element

Function: Request that a result set be no longer retained. A result set which is discarded, including CURRENT, becomes undefined. Subsequent attempts to reference it will cause a result set ID error.

Valid phases: Information Retrieval

Primitives and parameters:

<u>Parameter</u>	<u>Req/Ind</u>	<u>Res/Con</u>
Result_set_ID	[X]	
Discard_status		X

7.3.4 A_FETCH_RECORD Service Element

Function: To obtain a particular record identified by a record key without creating or changing any result sets. If a file name is supplied, it overrides the file name supplied as a subapplication parameter.

Valid phases: Information Retrieval

Primitives and parameters:

<u>Parameter</u>	<u>Req/Ind</u>	<u>Res/Con</u>
Fullness_code	X	
Record_key	X	
File_name	[X]	
Fetch_status		X
Data		[X]

7.4 Record Maintenance Facility

7.4.1 A_ADD_RECORD Service Element

Function: To add a new record to a file. If a file name is supplied, it overrides the file name supplied as a subapplication parameter. The record key may be supplied by the system making the ADD request; alternatively the receiving system may assign the record key, in which case it will be reported through the ADD_VALIDATED service element. If the requesting system has selected the validation mode, the responding system will assign a transaction reference number and return it in the response.

Valid phases: Record Maintenance Request

Primitives and parameters:

<u>Parameter</u>	<u>Req/Ind</u>	<u>Res/Con</u>
File_name	[X]	
Record_key	[X]	
Transaction_reference_number		[X]
Add_status		X
Data	X	

7.4.2 A_REPLACE_RECORD Service Element

Function: To cause a record in a file to be replaced. If a file name is supplied, it overrides the file name supplied as a subapplication parameter. If the requesting system has selected the validation mode, the responding system will assign a transaction reference number and return it in the response.

Valid phases: Record Maintenance Request

Primitives and parameters:

<u>Parameter</u>	<u>Req/Ind</u>	<u>Res/Con</u>
File_name	[X]	
Record_key	X	
Old_version_number	[X]	
Transaction_reference_number		[X]
Replace_status		X
Data	X	

7.4.3 A_DELETE_RECORD Service Element

Function: To request that a record be deleted from a file. If a file name is supplied, it overrides the file name supplied as a subapplication parameter. If the requesting system has selected the validation mode, the responding system will assign a transaction reference number and return it in the response.

Valid phases: Record Maintenance Request

Primitives and parameters:

<u>Parameter</u>	<u>Req/Ind</u>	<u>Res/Con</u>
File_name	[X]	
Record_key	X	
Old_version_number	[X]	
Transaction_reference_number		X
Delete_status		X

7.4.4 A_ADD_VALIDATED Service Element

Function: Indicate that a record has been validated and added to a file.

Valid phases: Record Maintenance Reporting

Primitives and parameters:

<u>Parameter</u>	<u>Req/Ind</u>	<u>Res/Con</u>
Add_validated_report	X	
Transaction_reference_number	X	
Record_key	X	
Diagnostic	[X]	
Validation_status		X
Data	[X]	

7.4.5 A_REPLACE_VALIDATED Service Element

Function: Indicate that a replacement record has been validated and the appropriate record has been replaced in the file.

Valid phases: Record Maintenance Reporting

Primitives and parameters:

<u>Parameter</u>	<u>Req/Ind</u>	<u>Res/Con</u>
Replace_validated_report	X	
Transaction_reference_number	X	
New_version_number	[X]	
Diagnostic	[X]	
Validation_status		X
Data	[X]	

7.4.6 A_DELETE_VALIDATED Service Element

Function: Indicate that a record has been deleted from a file.

Valid phases: Record Maintenance Reporting

Primitives and parameters:

<u>Parameter</u>	<u>Req/Ind</u>	<u>Res/Con</u>
Delete_validated_report	X	
Transaction_reference_number	X	
Validation_status		X

7.5 Mail Facility

7.5.1 A_SEND_MAIL Service Element

Function: Transmit mail from one end-user process to another end-user process located on a different host system. If mail notification is requested, either the originator or recipient must assign a mail reference number.

Valid phases: Mail Transmission

Primitives and parameters:

<u>Parameter</u>	<u>Req/Ind</u>	<u>Res/Con</u>
Mail_notification_mode	X	
Mail_reference_number	[X]	[X]
Recipient	X	
Mail_status		X
Mail_text	X	

7.5.2 A_MAIL_READ Service Element

Function: Indicate that mail has been read by the recipient.

Valid phases: Mail Read Notification

Primitives and parameters:

<u>Parameter</u>	<u>Req/Ind</u>	<u>Res/Con</u>
Mail_reference_number	X	
Mail_read_status		X

7.6 Interrupt Facility

7.6.1 A_CANCEL Service Element

Function: Allow the originating end-user process to terminate the processing of a request in the target end-user process.

Valid phases: Information Retrieval
Record Maintenance Request and Reporting
Mail Transmission and Read Notification

Primitives and parameters:

<u>Parameter</u>	<u>Req/Ind</u>
Cancel_reason	X

7.7 Settings Facility

7.7.1 A_SET Service Element

Function: Set subapplication parameters held by the peer. If any errors are detected, no settings are changed. Negotiation is not supported. If errors are detected, the response contains those parameters which correspond to the error code; all errors may not be reported by a given response.

Valid phases: Information Retrieval
Record Maintenance Request and Reporting
Mail Transmission and Read Notification

Primitives and parameters:

<u>Parameter</u>	<u>Req/Ind</u>	<u>Res/Con</u>
Subapplication_parameters	X	[X]
Set_status		X

7.7.2 A_SHOW Service Element

Function: Determine parameters held by peer.

Valid phases: Information Retrieval
Record Maintenance Request and Reporting
Mail Transmission and Read Notification

Primitives and parameters:

<u>Parameter</u>	<u>Req/Ind</u>	<u>Res/Con</u>
Service_options	[X] ¹	[X]

¹parameter name with zero length for value

8. PARAMETERS FOR SERVICE ELEMENTS

8.1 General

Abort reason: a code indicating the reason for abort, including an indication of whether the abort was caused by the local application entity, the remote application entity, or the remote application service user.

Account: the identification of the end-user process to the target system for charging and to determine privileges.

Add status: a code indicating the type of response to an add record request. Values:

- normal completion
- cancelled
- file name in error
- queue full

Add validated report: a code indicating the result of a record maintenance operation. Values:

- normal completion
- record added but unusual condition occurred (see diagnostic)
- duplicate record key (add not done)
- add rejected (see diagnostic)

Alternate query: a search request which a target end-user process generates when it is unable to perform the exact request made by the originator.

Called application: the identification of the application to which connection is being made.

Cancel reason: the reason for termination. Values:

- time out
- shutdown approaching
- terminal user request

Connection parameters: parameters which are specified by the originating end-user process at the start of a connection, and which may not be altered during the connection. They include:

- account
- user name
- password
- maximum length of data parameter
- timeout value

Data: one or more full or partial database records.

Delete status: a code indicating the type of response to a delete record request. Values:
normal completion
cancelled
file_name in error
queue full

Delete validated report: a code indicating the result of a record maintenance operation. Values:
normal completion
delete not allowed
record_key in error
version number incorrect

Diagnostic: a list of fields in error with a code indicating the type of error.

Discard status: a code indicating the type of response to a discard result set request. Values:
normal completion
cancelled
result_set_ID in error

Fetch status: a code indicating the type of response to a fetch request. Values:
normal completion
cancelled
fullness_code in error
file_name in error
record_key in error

File name: a character string which specifies the name of the file which is to be searched or maintained.

Fullness code: a code which indicates one of several possible levels of completeness of the records returned as the result of a retrieval request; e.g. count only, partial record, full record.

Mail notification mode: a code indicating whether the sender wishes to be notified when the recipient has read the mail. Values:
yes
no

Mail read status: a code indicating the type of response to a mail read request. Values:
normal completion
cancelled
mail_reference_number in error

Mail reference number: a number assigned by an end-user process to uniquely identify a particular mail request.

Mail status: a code indicating the type of response to a mail request. Values:

- normal completion
- cancelled
- recipient unknown
- mailbox full
- no mailbox exists
- message delivered; notification service not available

Mail text: the text of a message sent using the mail facility.

New version number: the version number assigned to a record which is replacing an existing record in a file.

Number of records: a count of the number of records requested from a result set or returned from a result set.

Old version number: the version number of an existing record which is to be replaced or deleted.

Password: a code maintained by the target system for the purpose of verifying that the entity using the account is authorized.

Present status: a code indicating the type of response to a present records request. Values:

- normal completion
- cancelled
- no current result set
- result_set_ID in error
- fullness_code in error
- number_of_records out of bounds
- result_set_position out of bounds
- number_of_records too large for application

Query: A search query. See section 8.2.

Recipient: the individual to whom a message is addressed.

Record format: A code indicating one of the possible record formats. Values (for example):

- U.S. MARC
- Canadian MARC

Record key: an identifier which uniquely specifies a record within a file.

Record maintenance validation mode: A code indicating whether record validation is required. Values:

- yes
- no

Release reason: a code indicating the reason (e.g. normal completion) that the end-user process released the connection.

Replace status: a code indicating the type of response to a replace record request. Values:

- normal completion
- cancelled
- file_name in error
- queue full

Replace validated report: a code indicating the result of a record maintenance operation. Values:

- normal completion
- record replaced but unusual condition occurred (see diagnostic)
- record_key in error
- version number incorrect
- replace rejected (see diagnostic)

Result count: The number of records in a result set.

Result set ID: An identifier assigned to a result set from a search. It may be used in subsequent requests for records from that set. The identifier is assigned by the target end-user process. The name CURRENT is reserved, and is always used to identify the results of the most recent search in the same session which created a non-null result set. The name ALL is reserved, and is used to identify all existing result sets in a DISCARD_RESULT request.

Result set position: An integer used to specify the position of a particular record within a result set. The first record in a result set is record 1.

Search diagnostic: a search query with an indication of the portions in error.

Search status: a code indicating the type of response to a search request. Values:

- normal completion
- cancelled
- form of query not supported
- form of query not supported, alternate performed
- result set too large to process

invalid subapplication parameter
time limit exceeded
query in error (see search_diagnostic)

Service options: A group of options which apply to a particular end-user process. These include:
files available
search types supported
mail recipients

Set status: a code indicating the response to a set request.
Values:
normal completion
cancelled
subapplication parameter unknown
subapplication parameter value invalid

Statistics: Identification and values of certain connection statistics, such as:
number of messages
elapsed time

Subapplication ID: a code which indicates the subapplication.
Values:
information retrieval
record maintenance requests
record maintenance reporting
mail
mail read notification

Subapplication parameters: Parameters which apply to a particular subapplication and which may be changed by the originating end-user process. They include:
file name (logical file name)
small result count (upper bound to number of records to be considered a small result)
small result fullness code
medium result count (upper bound to number of records to be considered a medium result)
medium result fullness code
large result count (upper bound to number of records to be considered a large result)
large result fullness code
search time limit
result set disposition
record maintenance validation mode
record format

Transaction reference number: A number assigned by the target end-user process to uniquely identify a particular file maintenance request.

Validation status: a code indicating the response to a validation request. Values:
normal completion
cancelled
unrecognized transaction reference number

8.2 Formal Description of the Search Query

The following description serves to describe the syntax which is supported by the application service to convey search queries. This description shows the possible constructs. It does not try to show what is not possible. It is a description of syntax, not of semantics.

The following symbols have special uses in the BNF (Backus Normal Form) formalism:

::= meaning "is defined as"
< > enclosing terms which are further defined later in the description
UPPER_CASE is used for terminal symbols
[] enclosing terms which may occur optionally zero or more times
(* *) enclosing explanatory comments
OR meaning "or"
... indicating that not all valid terminal symbols in a sequence have been shown

<query> ::= <file name> <query structure>

<file name> ::= <simple character string> OR DEFAULT

<query structure> ::= RPN <RPN structure>
OR (* others (for future extension) *)

<RPN structure> ::= <term> [[<term>] [<logical operator>]]

<term> ::= TERM <term value> USE <use value>
RELATION <relation value> [<optional attribute>]

<term value> ::= <extended character string>

<optional attribute> ::= POSITION <position value>
OR STRUCTURE <structure value>
OR TRUNCATION <truncation value>


```

<position value> ::= FIRST_IN_FIELD
                     OR FIRST_IN_ANY_SUBFIELD
                     OR FIRST_IN_SUBFIELD_1a
                     OR FIRST_IN_NOT_SUBFIELD_1a
                     OR ANY_POSITION

<relation value> ::= EQUAL
                     OR GREATER_THAN
                     OR GREATER_THAN_OR_EQUAL
                     OR LESS_THAN
                     OR LESS_THAN_OR_EQUAL
                     OR NOT_EQUAL

<structure value> ::= PHRASE (* includes personal name *)
                     OR WORD_LIST (* includes single word *)
                     OR 3,3_KEY
                     OR 3,1,1,1_KEY
                     OR (* others *)

<truncation value> ::= NO_TRUNCATION
                      OR RIGHT_TRUNCATION

<use value> ::= ANY
               OR AUTHOR
               OR AUTHOR/TITLE
               OR CONFERENCE_NAME
               OR CORPORATE_NAME
               OR ISBN
               OR ISSN
               OR LC_CARD_NUMBER
               OR NOTE
               OR PERSONAL_NAME
               OR RESULT_SET_ID
               OR SERIES
               OR SUBJECT
               OR TITLE
               OR UNIFORM_TITLE
               OR 001_FIELD
               OR 035_FIELD
               OR (* others *)

<logical operator> ::= AND
                     OR AND_NOT
                     OR NOT
                     OR OR

<simple character string> ::= 0 OR 1 OR ... 9 OR A OR B
                             OR ... Z [<simple character
                             string>]

```

<extended character string> ::= <MARC/ALA> [<MARC/ALA>]

<MARC/ALA> ::= (* any graphic character defined in the MARC/ALA
8-bit extended character set *)

9. SPECIFICATION OF INTERFACE EVENT SEQUENCES

This section illustrates the valid sequences of interface events as seen by a user of the application service at one end of an application connection. It includes both the events initiated by the user and those initiated by the application service. The 'normal' or most common sequence is shown on a single horizontal line from left to right. There is no precedence signified by the vertical ordering of events. After completing a sequence from the left to the right margins, all sequences starting at the left are again available unless a path has been taken which led to some other phase as indicated by the square brackets.

key:

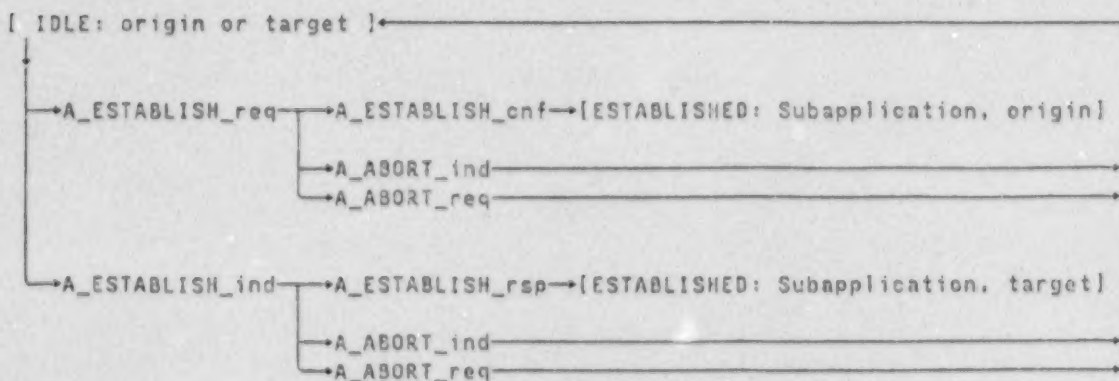
[] = name of a phase (cf Section 3.3, Figure 2)

Events: (cf Section 7)

req = request (from local user to application service)
ind = indication (to local user from application service)
rsp = response (from local user to application service)
cnf = confirm (to local user from application service)

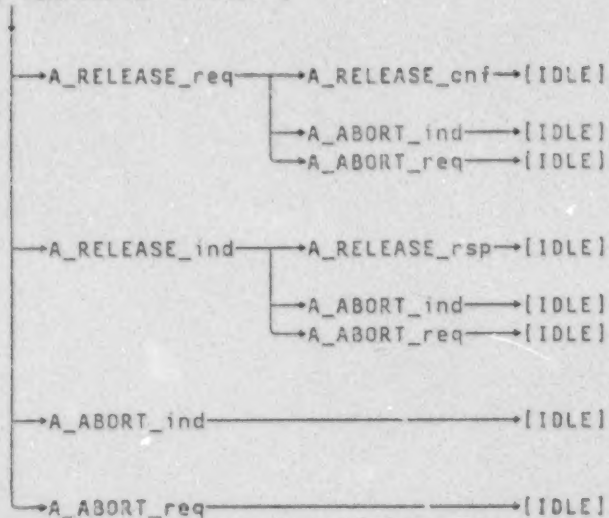
9.1 CONNECTION

9.1.1 Establishment

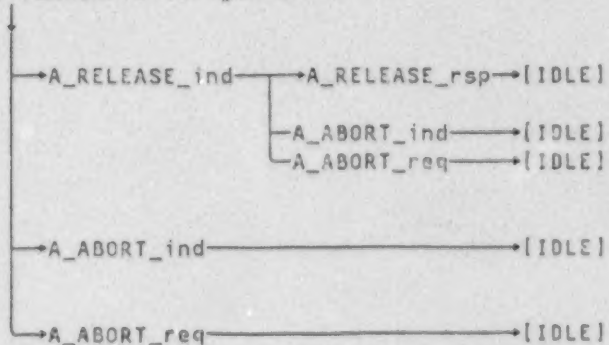


9.1.2 Termination

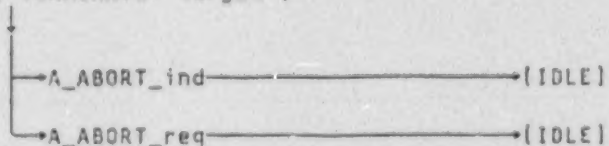
[TERMINATE: either]



[TERMINATE: origin]

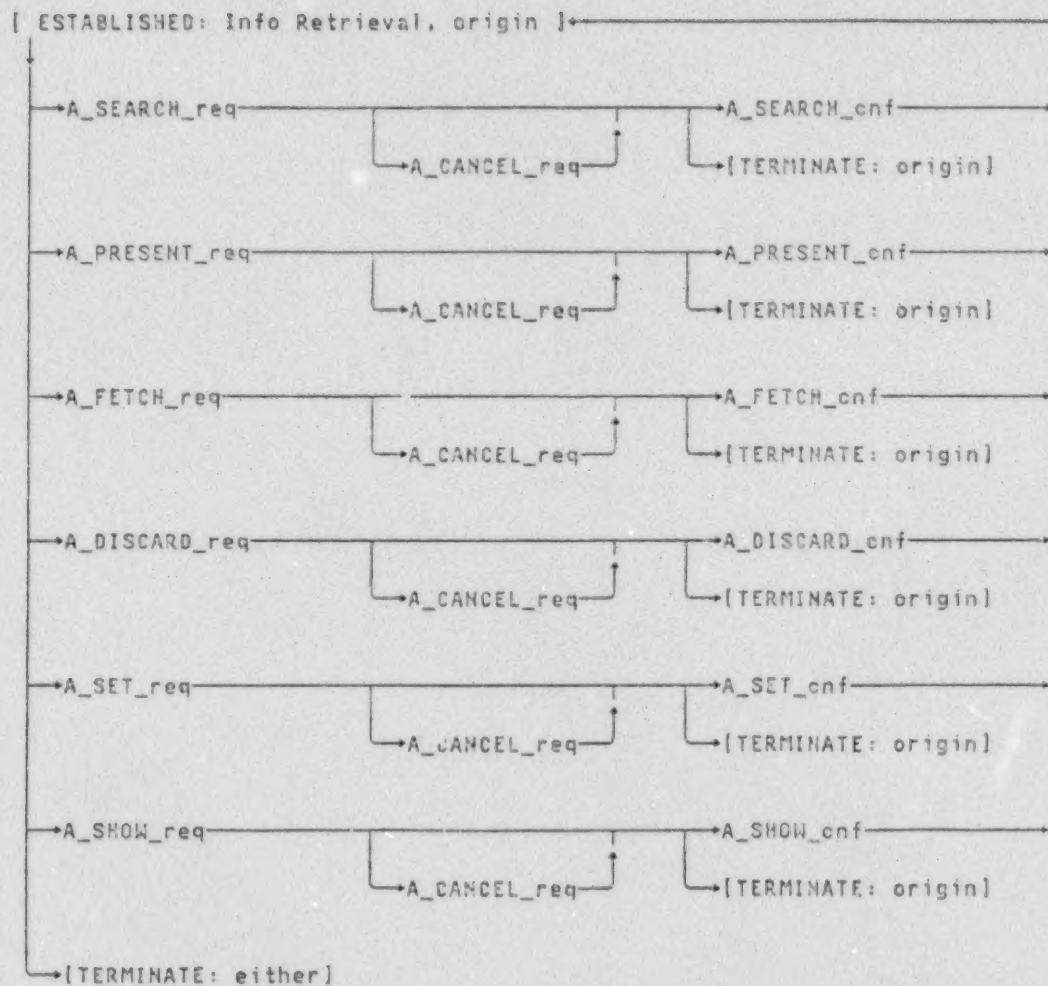


[TERMINATE: target]

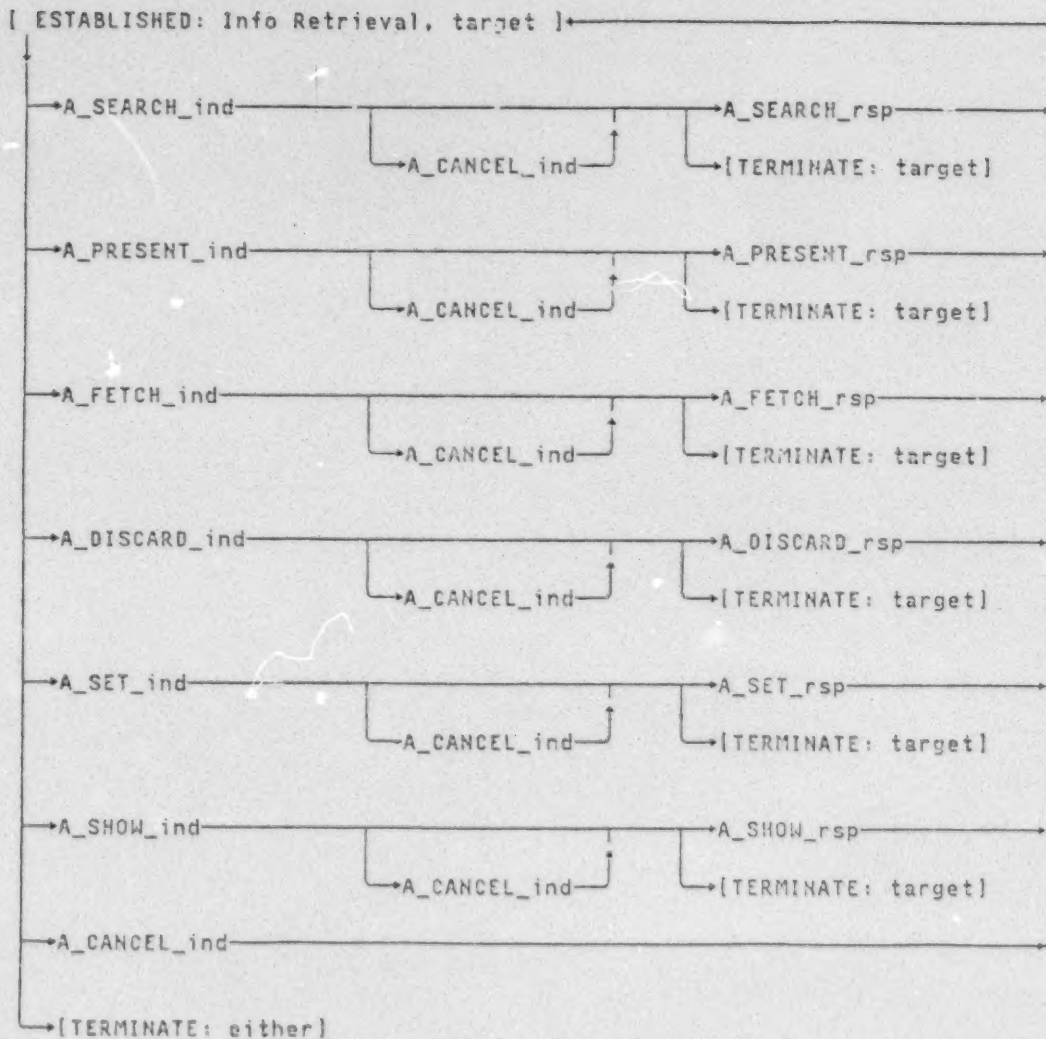


9.2 INFORMATION RETRIEVAL

9.2.1

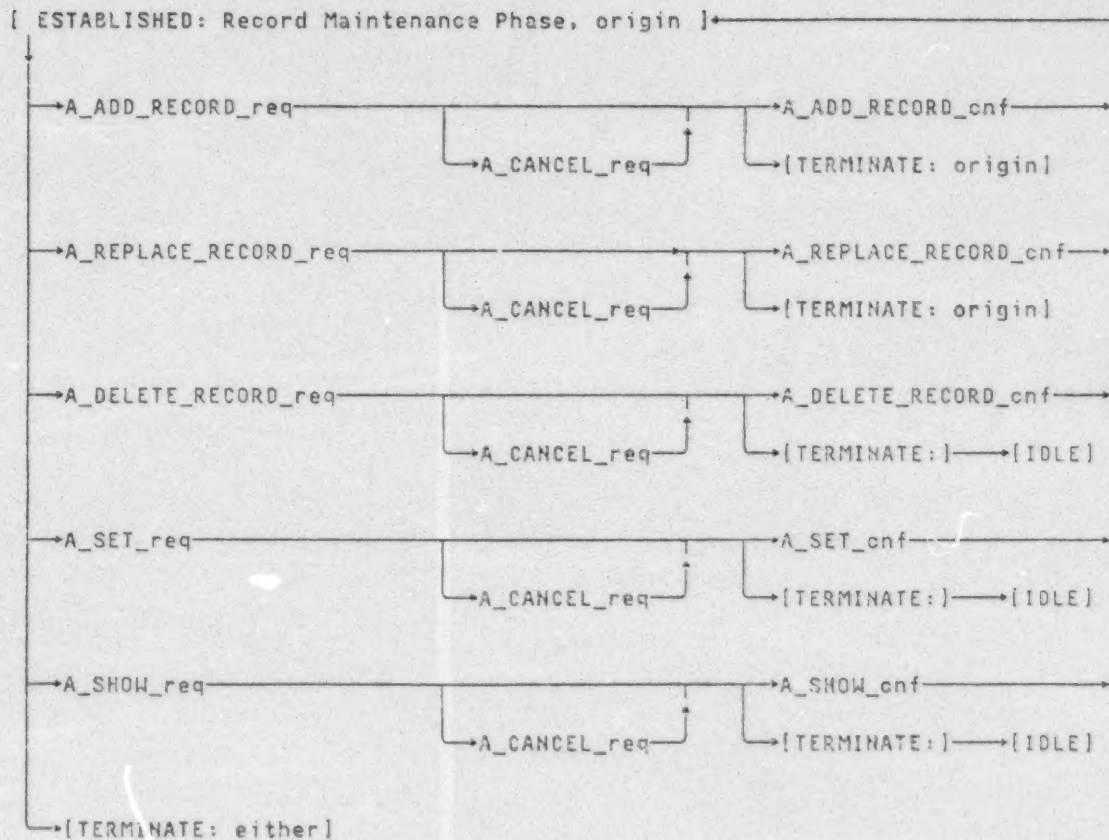


9.2.2

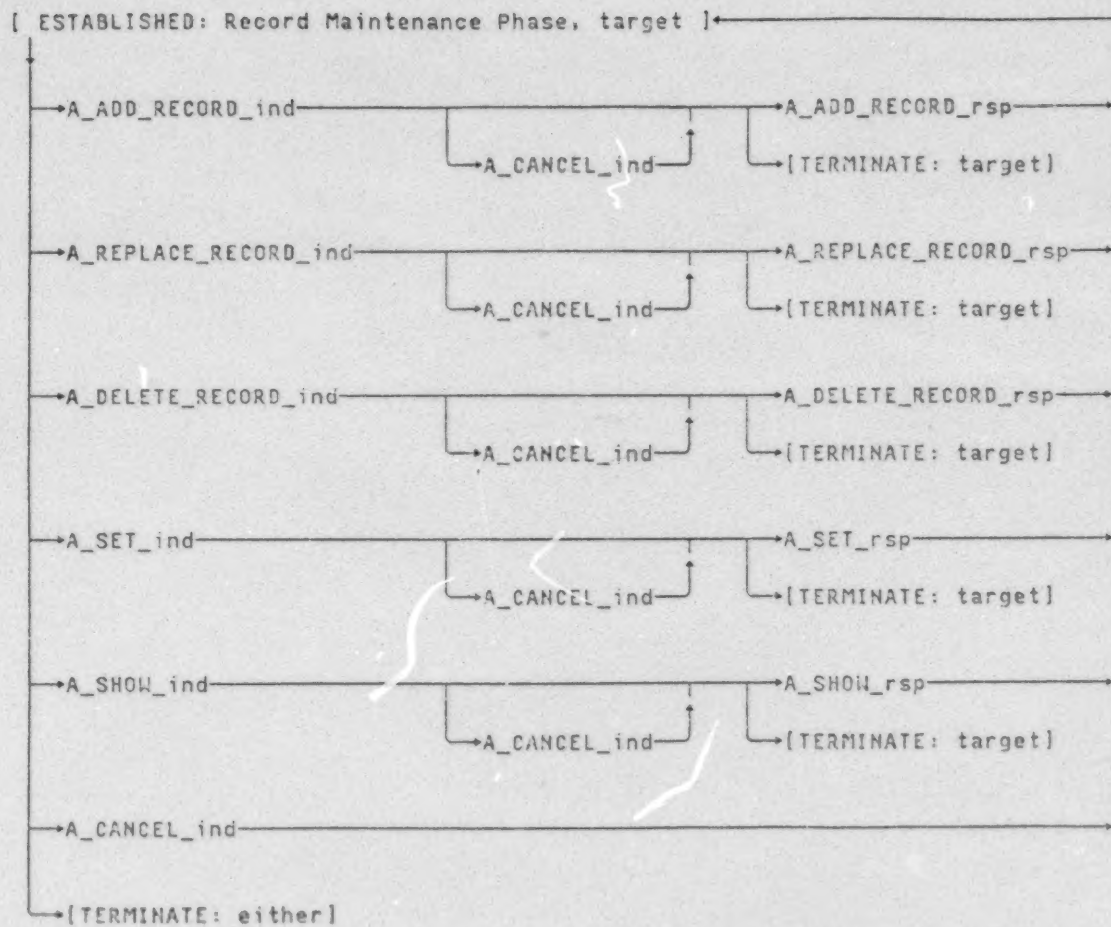


9.3 RECORD MAINTENANCE

9.3.1



9.3.2

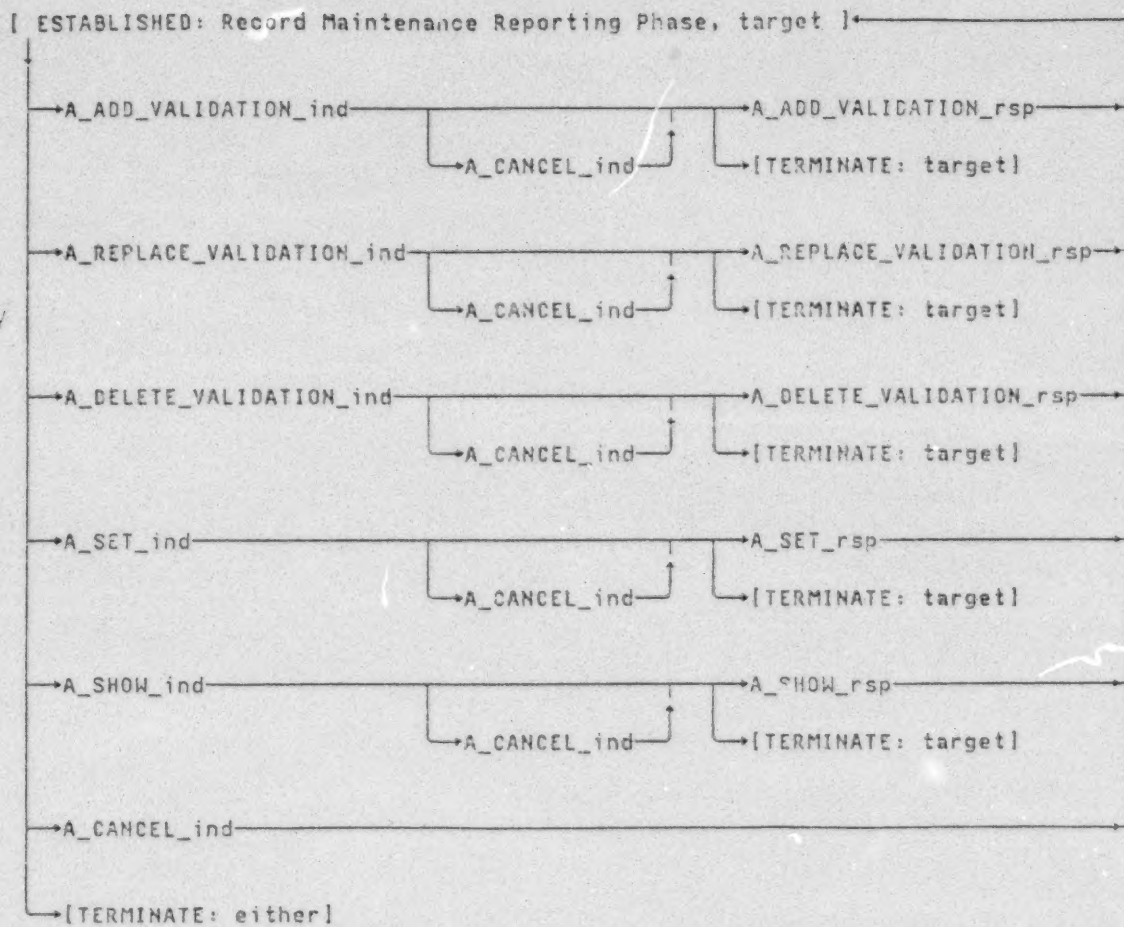


```

sequenceDiagram
    participant A as [ ESTABLISHED: Record Maintenance Reporting Phase, origin ]
    A->>A: A_ADD_VALIDATION_req
    A->>A: A_ADD_VALIDATION_cnf
    A->>A: A_CANCEL_req
    A->>A: [TERMINATE: origin]
    A->>A: A_REPLACE_VALIDATION_req
    A->>A: A_REPLACE_VALIDATION_cnf
    A->>A: A_CANCEL_req
    A->>A: [TERMINATE: origin]
    A->>A: A_DELETE_VALIDATION_req
    A->>A: A_DELETE_VALIDATION_cnf
    A->>A: A_CANCEL_req
    A->>A: [TERMINATE: origin]
    A->>A: A_SET_req
    A->>A: A_SET_req
    A->>A: A_CANCEL_req
    A->>A: [TERMINATE: origin]
    A->>A: A_SHOW_req
    A->>A: A_SHOW_cnf
    A->>A: A_CANCEL_req
    A->>A: [TERMINATE: origin]
    A->>A: [TERMINATE: either]

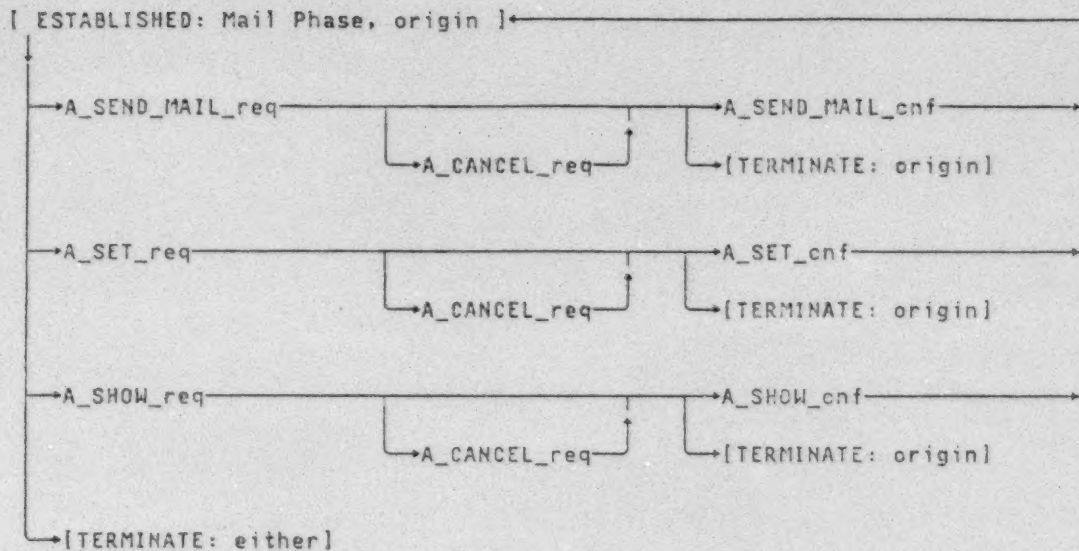
```


9.3.4

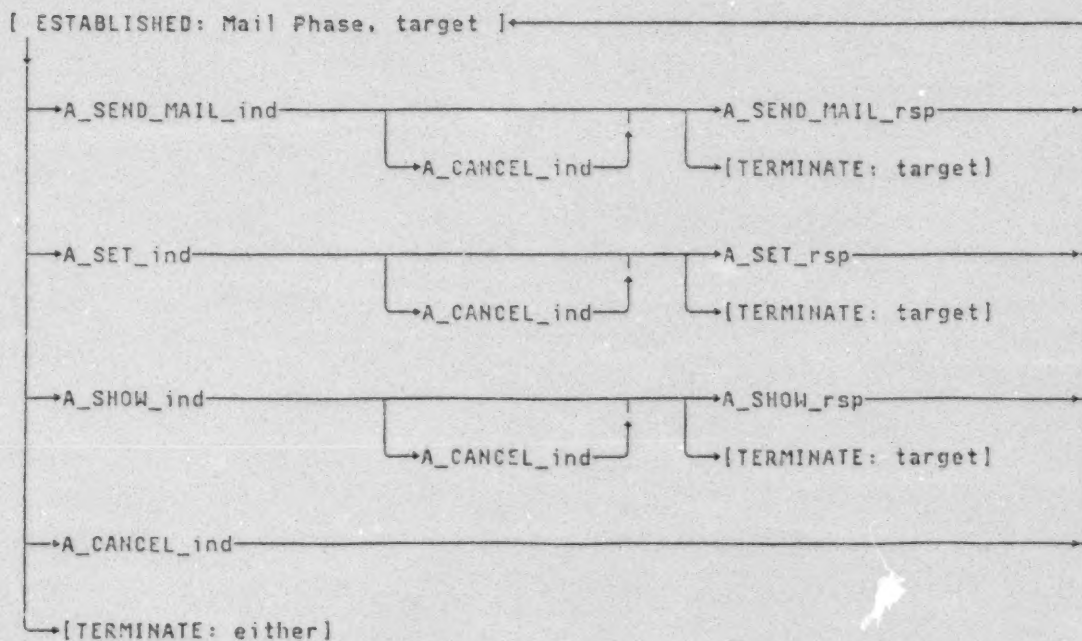


9.4 MAIL

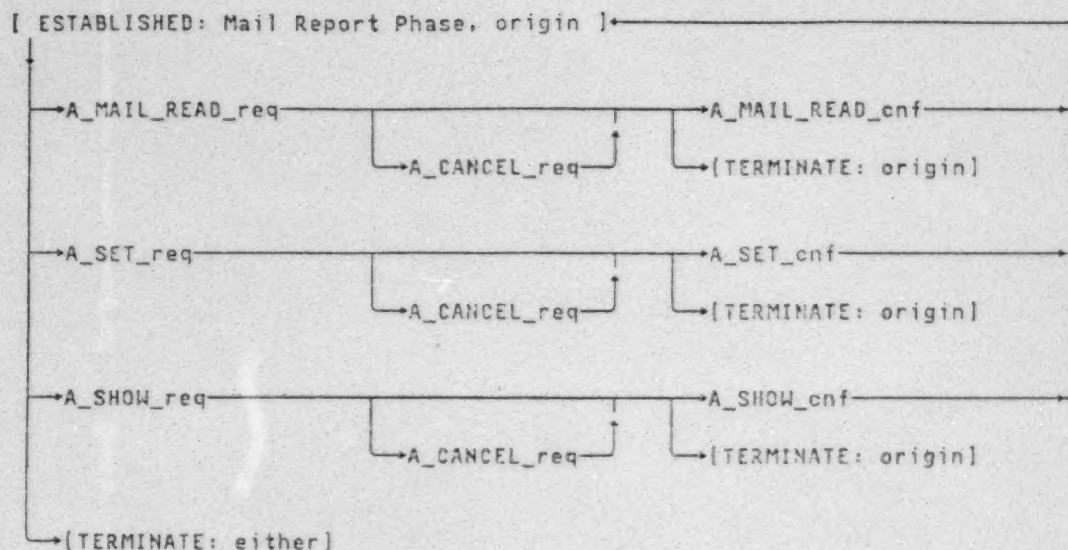
9.4.1



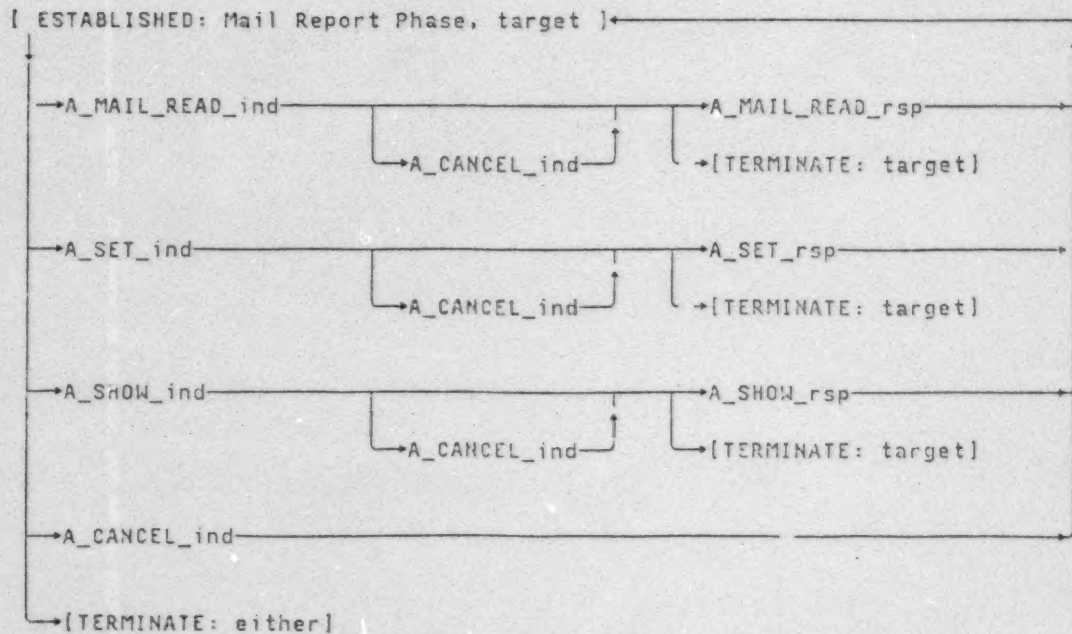
9.4.2



9.4.3



9.4.4



REFERENCES

- [BBN-81/02] Bolt Beranek and Newman, Inc. An Automated Formal Specification Technique for Protocols. Cambridge, Mass., February, 1981. Working paper.
- [BCL-80/09] Batelle Columbus Laboratories. Linking the Bibliographic Utilities: Benefits and Costs. Columbus, Ohio, September 15, 1980. Technical Report Submitted to the Council on Library Resources.
- [BIBNETT] Steering Committee of BIBNETT. Progress report from phase 3. "An application protocol for library and information systems". Oslo, Norway, March 13, 1981.
- [BIG-82/04] Computer Gateways, Inc. A File Transfer Service for the Bibliographic Interest Group. Ottawa, Canada, April 30, 1982. Draft.
- [CBOS-80-02] United States. Dept. of Commerce. National Bureau of Standards. Specification of a Draft Message Format Standard. October, 1980. Report no. ICST/CBOS-80-2.
- [CBOS-82-01] United States. Dept. of Commerce. National Bureau of Standards. Features of a Message Transfer Protocol. November, 1981. Report no. ICST/CBOS-82-1.
- [CBOS-82-03] United States. Dept. of Commerce. National Bureau of Standards. Service Specification of a Message Transfer Protocol. February, 1982. Report no. ICST/CBOS-82-3.
- [DP7498] International Organisation for Standardisation. Data Processing - Open Systems Interconnection - Basic Reference Model. December 3, 1980. Report no. ISO/DP7498, replaces ISO/TC97/SC16/N227.
- [FIPS-82/04] United States. Dept. of Commerce. National Bureau of Standards. Specification for Message Format for Computer Based Message Systems. April, 1982. Proposed Federal Information Processing Standard.
- [HLNP-80-01] United States. Dept. of Commerce. National Bureau of Standards. Features of the Transport and Session Protocols. March, 1980. Report no. ICST/HLNP-80-1.

- [HLNP-80-02] United States. Dept. of Commerce. National Bureau of Standards. Service Specification of Transport and Session Protocols. March, 1980. Report no. ICST/HLNP-80-2.
- [HLNP-80-03] United States. Dept. of Commerce. National Bureau of Standards. Formal Description Techniques for Network Protocols. June, 1980. Report no. ICST/HLNP-80-3.
- [HLNP-80-04] United States. Dept. of Commerce. National Bureau of Standards. Common Command Language Feature Analysis. June, 1980. Report no. ICST/HLNP-80-4.
- [HLNP-80-06] United States. Dept. of Commerce. National Bureau of Standards. Features of the File Transfer Protocol (FTP) and the Data Presentation Protocol (DPP). September, 1980. Report no. ICST/HLNP-80-6.
- [HLNP-80-07] United States. Dept. of Commerce. National Bureau of Standards. Formal Methods for Communication Protocol Specification and Verification. June, 1980. Report no. ICST/HLNP-80-7.
- [HLNP-80-08] United States. Dept. of Commerce. National Bureau of Standards. Features of Internetwork Protocol. July, 1980. Report no. ICST/HLNP-80-8.
- [HLNP-80-09] United States. Dept. of Commerce. National Bureau of Standards. Service Specification of the File Transfer Protocol (FTP) and the Data Presentation Protocol (DPP). October, 1980. Report no. ICST/HLNP-80-9.
- [HLNP-80-11] United States. Dept. of Commerce. National Bureau of Standards. Service Specification of an Internetwork Protocol. September, 1980. Report no. ICST/HLNP-80-11.
- [HLNP-80-12] United States. Dept. of Commerce. National Bureau of Standards. Features of Network Interprocess Communication Protocols. September, 1980. Report no. ICST/HLNP-80-12.
- [HLNP-80-15] United States. Dept. of Commerce. National Bureau of Standards. Service Specification of a Network Interprocess Communication Protocol. October, 1980. Report no. ICST/HLNP-80-15.

- [HLNP-81-02] United States. Dept. of Commerce. National Bureau of Standards. Specification of the Session Protocol. Revised March, 1981. Report no. ICST/HLNP-81-2.
- [HLNP-81-04] United States. Dept. of Commerce. National Bureau of Standards. Specification of the File Transfer Protocol. December, 1981. Report no. ICST/HLNP-81-4.
- [HLNP-81-05] United States. Dept. of Commerce. National Bureau of Standards. Specification of the Data Presentation Protocol. December, 1981. Report no. ICST/HLNP-81-5.
- [HLNP-81-07] United States. Dept. of Commerce. National Bureau of Standards. Feature Analysis of the Integrated Upper Levels Protocol. May, 1981. Report no. ICST/HLNP-81-7.
- [HLNP-81-08] United States. Dept. of Commerce. National Bureau of Standards. Service Features of the Integrated Upper Levels Protocol. May, 1981. Report no. ICST/HLNP-81-8.
- [HLNP-81-11] United States. Dept. of Commerce. National Bureau of Standards. Specification of the Transport Protocol, Volume 1: Overview and Services. September, 1981. Report no. ICST/HLNP-81-11.
- [HLNP-81-12] United States. Dept. of Commerce. National Bureau of Standards. Specification of the Transport Protocol, Volume 2: Basic Class Protocol. September, 1981. Report no. ICST/HLNP-81-12.
- [HLNP-81-13] United States. Dept. of Commerce. National Bureau of Standards. Specification of the Transport Protocol, Volume 3: Extended Class Protocol. September, 1981. Report no. ICST/HLNP-81-13.
- [HLNP-81-14] United States. Dept. of Commerce. National Bureau of Standards. Specification of the Transport Protocol, Volume 4: Network Interfaces. September, 1981. Report no. ICST/HLNP-81-14.
- [HLNP-81-15] United States. Dept. of Commerce. National Bureau of Standards. A Formal Specification Technique and Implementation Method for Protocols. July, 1981. Report no. ICST/HLNP-81-15.

- [HLNP-81-17] United States. Dept. of Commerce. National Bureau of Standards. The Impact of Satellite Transmission on High-level Computer Network Protocols. August, 1981. Report no. ICST/HLNP-81-17.
- [HLNP-81-18] United States. Dept. of Commerce. National Bureau of Standards. The Effects of Satellite Technology on the ISO Model of Open Systems Interconnection. August, 1981. Report no. ICST/HLNP-81-18.
- [HLNP-81-19] United States. Dept. of Commerce. National Bureau of Standards. Security in Higher Level Protocols: Approaches, Alternatives, and Recommendations. September, 1981. Report no. ICST/HLNP-81-19.
- [HLNP-81-20] United States. Dept. of Commerce. National Bureau of Standards. A Benchmark for Implementations of the NBS Transport Protocol. September, 1981. Report no. ICST/H[L]NP-81-20.
- [IEEE-80/04] The Institute of Electrical and Electronics Engineers, Inc. Communications Society. IEEE Transactions on Communications. "Special Issue on Computer Network Architectures and Protocols", April, 1980 (Volume COM-28, number 4). ISSN 0090-6778.
- [KVO-81] Kuo, Franklin F. Protocols and Techniques for Data Communication Networks. Englewood Cliffs, N.J., Prentice-Hall, 1981. ISBN 0-13-731729-8.
- [LASP-39] Washington Library Network, The Research Libraries Group, Inc., and Library of Congress. Linked Authorities Systems Project. Authorities Group. LSP External Design: Intersystem Component. February 26, 1982. Report no. LASP-39.
- [MFBD] United States. Library of Congress. Automated Systems Office. MARC Formats for Bibliographic Data. March 25, 1980. ISBN 0-8444-0333-4. Updated quarterly.
- [NTAG-77-80] NCLIS/NBS Task Force on Computer Network Protocol. Final Report and Recommendations. August, 1977. Second draft.
- [NTAG-78-21] Protocol Implementation Planners. Computer-to-Computer Communications Protocol Implementation Plan for the National Library and Information Service Network. New York, August 2, 1978. Final report to the Library of Congress Network Development Office.

- [NTAG-78-22] INSPEC, The Institution of Electrical Engineers.
EURONET Guideline: Standard Commands for Retrieval Systems. London, England, December, 1977. Final report on a study carried out for the Commission of the European Communities, DG XIII.
- [N0016] International Organisation for Standardisation.
Working Draft for Search and Operational Support Protocols (Command Language for Interrogation of Information Retrieval Systems). December, 1980. Report no. ISO/TC46/SC4/WG5/N16, replaces ISO/TC46/SC4/WG5/N13.
- [N0856] International Organisation for Standardisation. Draft Basic Connection-oriented Session Protocol Specification. January, 1982. Report no. ISO/TC97/SC16/N856.
- [N0898] International Organisation for Standardisation.
Guidelines for the Specification of Services for OSI. January, 1982. Report no. ISO/TC97/SC16/N898, replaces ISO/TC97/SC16/N577.
- [N0899] International Organisation for Standardisation.
Guidelines for the Specification of Protocols for OSI. January, 1982. Report no. ISO/TC97/SC16/N899, replaces ISO/TC97/SC16/N577.
- [N0918] International Organisation for Standardisation. Issues Pertinent to Application and Presentation Layers. March, 1982. Report no. ISO/TC97/SC16/N918.
- [N1161] International Organisation for Standardisation. Draft Connection Oriented Presentation Service Definition. June, 1982. Report no. ISO/TC97/SC16/N1161, replaces ISO/TC97/SC16/N919.
- [N1162] International Organisation for Standardisation.
Information Processing Systems - Open Systems Interconnection - Transport Service Definition. June, 1982. Report no. ISO/TC97/SC16/N1162.
- [N1169] International Organisation for Standardisation.
Information Processing Systems - Open Systems Interconnection - Transport Protocol Specification. June, 1982. Report no. ISO/TC97/SC16/N1169.

- [N1190] International Organisation for Standardisation.
Working Draft on File Transfer, Access and Management -
June 1982 - General Description. Report no.
ISO/TC97/SC16/N1190.
- [N1222] International Organisation for Standardisation.
Working Draft on File Transfer, Access and Management -
June 1982 - The Virtual Filestore. Report no.
ISO/TC97/SC16/N1222.
- [N1223] International Organisation for Standardisation.
Working Draft on File Transfer, Access and Management -
June 1982 - The File Service Definition. Report no.
ISO/TC97/SC16/N1223.
- [N1224] International Organisation for Standardisation.
Working Draft on File Transfer, Access and Management -
June 1982 - The File Protocol Specification. Report
no. ISO/TC97/SC16/N1224.
- [RFC-733] United States. Department of Defense. Advanced
Research Projects Agency. Committee on Computer-aided
Human Communication. Standard for the Format of ARPA
Network Text Messages. November 21, 1977. Request for
comments no. 733.
- [RR-80-43] International Institute for Applied Systems Analysis.
Remote File Access: A Data-access Protocol for
Computer Networks. Laxenburg, Austria, December, 1980.
Report no. RR-80-43. NTIS publication no. PB81-159840.
- [TAN-81] Tanenbaum, Andrew S. Computer Networks. Englewood
Cliffs, N.J., Prentice-Hall, 1981. ISBN 0-13-165183-8.
- [X3T5.5/82-121] American National Standards Institute.
Subcommittee X3T5.5. Rapporteur's Group on Upper Layer
Architecture. Common Application Layer Service Element
Definition. July 30, 1982. Report no. X3T5.5/82-121.
- [X3T5/82-209] Nielsen, F. Analysis of Standards Efforts -
Messaging and its Relationship to the OSI. Undated
letter to ANSI subcommittee X3T5. Report no.
X3T5/82-209.
- [Z39] Anonymous. Search and Retrieval in a
Computer-to-Computer Environment. Undated.
Unnumbered.

U.S. DEPARTMENT OF EDUCATION
NATIONAL INSTITUTE OF EDUCATION
EDUCATIONAL RESOURCES INFORMATION
CENTER (ERIC)

This document has been reproduced as
received from the person or organization
originating it.
Minor changes have been made to improve
reproduction quality.

- Points of view or opinions stated in this docu-
ment do not necessarily represent official NIE
position or policy.

Report no. TG.50

Application Level Protocol Development
for
Library and Information Science Applications
Volume 2
Protocol Specification
Council on Library Resources
Grant 2033

Prepared by:
James S. Aagaard
Kenton E. Andersen
Wayne E. Davison

October 29, 1982

PERMISSION TO REPRODUCE THIS
MATERIAL HAS BEEN GRANTED BY

Council on Library
Resources

TO THE EDUCATIONAL RESOURCES
INFORMATION CENTER (ERIC).

TABLE OF CONTENTS

10.	INTRODUCTION	1
11.	OVERVIEW OF THE PROTOCOL	2
11.1	General	2
11.2	Services Provided by the Application Layer	2
11.3	Services Required from the Presentation Layer	4
11.4	Services Required from the Operating System	5
11.5	Examples of Protocol Operation	5
11.5.1	Connection Establishment	6
11.5.2	Information Retrieval	10
11.5.3	Record Maintenance	14
11.5.4	Mail Transmission	14
11.5.5	Connection Termination	14
12.	PROTOCOL SPECIFICATION METHOD AND NOTATION	21
12.1	General	21
12.2	The Machine Model	21
12.3	Description of the Specification Technique	22
12.4	Special Notational Devices	24
12.4.1	Transition Abbreviation	24
12.4.2	Predicate Abbreviation	25
12.4.3	Primitive Procedures	25
12.4.4	Notation for Labels and Special Purposes	26
12.4.5	Presentation Layer Object List Notation	26
13.	FORMAL SPECIFICATION OF THE PROTOCOL	28
13.1	Interface Definitions	28

13.2	State Descriptions	28
13.3	Local Machine Variables	33
13.4	Primitive Procedures	34
13.5	Initialization of the Machine	35
13.6	State Transition Definitions	36
14.	PROTOCOL DATA UNIT FORMATS	115
14.0	Introduction	115
14.0.1	Notation	115
14.0.2	Basic PDU Format	116
14.0.3	Basic Header Format	116
14.1	PDU Data Elements	117
14.2	PDU Formats	123
14.2.1	Establish Connection PDU	123
14.2.2	Accept Connection PDU	123
14.2.3	Abort PDU	123
14.2.4	Release Connection PDU	124
14.2.5	Release Answer PDU	124
14.2.6	Set PDU	124
14.2.7	Set Response PDU	124
14.2.8	Show PDU	125
14.2.9	Show Response PDU	125
14.2.10	Cancel PDU	125
14.2.11	Error PDU	126
14.2.12	Add Record PDU	126
14.2.13	Add Received PDU	127
14.2.14	Replace Record PDU	127

14.2.15	Replace Received PDU	128
14.2.16	Delete Record PDU	128
14.2.17	Delete Received PDU	128
14.2.18	Add Validated PDU	129
14.2.19	Replace Validated PDU	129
14.2.20	Delete Validated PDU	130
14.2.21	Validation Ack PDU	130
14.2.22	Search PDU	131
14.2.23	Search Response PDU	132
14.2.24	Present Records PDU	133
14.2.25	Presented Records PDU	133
14.2.26	Discard Result Set PDU	134
14.2.27	Result Set Discarded PDU	134
14.2.28	Fetch Record PDU	135
14.2.29	Fetch Record PDU	135
14.2.30	Send Mail PDU	136
14.2.31	Mail Response PDU	136
14.2.32	Message Read PDU	137
14.2.33	Message Read Response PDU	137
REFERENCES		138

10. INTRODUCTION

The protocol specified in this document resides in the upper layer of the Reference Model for Open Systems Interconnection (OSI) [DP7498].¹ It implements the features necessary for bibliographic searching, record maintenance, and mail transfer between end-user application processes in library and information sciences.

In the OSI model a layer only provides services to the layer immediately above it, and uses the services of the layer immediately below it. In the case of the application layer, the "layer immediately above" is the end-user process.

An entity is a process or module that provides the services required of the layer within a single system. Entities in the same layer are called "peer entities." Peer entities communicate according to a set of rules. This set of rules is called the "protocol" of the layer.

A protocol provides for a predefined set of procedures and formats. Through this protocol, peer entities can control the flow of information. A unit of data, in a predefined format, passed between two peer entities, is called a "protocol data unit" or "PDU."

Control information and data will be passed to the peer entities using the services of the lower layers. The lower layers provide a virtual link between the peer entities. Communication actually proceeds down through the lower layers, across the physical link, and up through the layers until it reaches the peer entity of the associated layer. To accomplish this, each layer provides for the delivery of the PDUs of the next higher layer.

¹The notation [...] is used in this document to reference an item in the bibliography.

11. OVERVIEW OF THE PROTOCOL

11.1 General

The application protocol which is described in this report provides several types of services: connection management, information retrieval, record maintenance, and mail. These are further subdivided into five subapplications, as follows:

- Information retrieval
- Record maintenance requests
- Record maintenance reporting
- Mail transmission
- Mail read notification

The originating application selects the subapplication at the time the connection is established. Switching between subapplications during a connection is not supported.

11.2 Services Provided by the Application Layer

A complete definition of the services provided by the application layer to the end-user process is given in Volume 1 of this report. The services are summarized here for convenience. Note that the parameter column contains all of the parameters which may be used by a given service element. A given primitive may use only a subset of these, and some parameters may be optional.

<u>Service Element</u>	<u>Primitive</u>	<u>Parameters</u>
A_ESTABLISH	request	Called_application
	indication	Connection_parameters
	response	Subapplication_ID
	confirm	Subapplication parameters
A_ABORT	request	Abort_reason
	indication	
A_RELEASE	request	Release_reason
	indication	Statistics
	response	
	confirm	
A_SET	request	Subapplication_parameters
	indication	Set_status
	response	
	confirm	

A_SHOW	request indication response confirm	Service_options
A_CANCEL	request indication	Cancel_reason
A_ADD_RECORD	request indication response confirm	File_name Record_key Transaction_reference_number Add_status Data
A_REPLACE_RECORD	request indication response confirm	File_name Record_key Old_version_number Transaction_reference_number Replace_status Data
A_DELETE_RECORD	request indication response confirm	File_name Record_key Old_version_number Transaction_reference_number Delete_status
A_ADD_VALIDATED	request indication response confirm	Add_validated_report Transaction_reference_number Record_key Diagnostic Validation_status Data
A_REPLACE_VALIDATED	request indication response confirm	Replace_validated_report Transaction_reference_number New_version_number Diagnostic Validation_status Data
A_DELETE_VALIDATED	request indication response confirm	Delete_validated_report Transaction_reference_number Validation_status

A_SEARCH	request indication response confirm	RPN_query Search_status Result_count Result_set_ID Fullness_code Search_diagnostic Alternate_query Number_of_records Data
A_PRESENT_RECORDS	request indication response confirm	Fullness_code Result_set_position Number_of_records Result_set_ID Data Present_status
A_DISCARD_RESULT	request indication response confirm	Result_set_ID Discard_status
A_FETCH_RECORD	request indication response confirm	Fullness_code Record_key File_name Fetch_status Data
A_SEND_MAIL	request indication response confirm	Mail_notification_code Mail_reference_number Recipient Mail_status Mail_text
A_MAIL_READ	request indication response confirm	Mail_reference_number Mail_read_status

11.3 Services Required from the Presentation Layer

This section summarizes the services which are required from the presentation layer. A presentation layer which provides these services is specified in the companion report Presentation Level Protocol Development for Library and Information Science Applications, to which reference should be made for a complete definition of the services and parameters.

<u>Service Element</u>	<u>Primitive</u>	<u>Parameters</u>
P_CONNECT	request indication response confirm	Initiator_address Acceptor_address Initial_context_name Initial_neg_right Context_set User-data
P_U_ABORT	request indication	User_data
P_P_ABORT	indication	Reason
P_DATA	request indication	Operation Object_list
P_EXP_DATA	request indication	Operation Object_list

11.4 Services Required from the Operating System

The following services are assumed to be supplied by the operating system environment. (However, none are used in the current version of the protocol.)

<u>Service Element</u>	<u>Primitive</u>	<u>Parameters</u>
E_TIMER_SET	request indication	Timer_name Timer_value
E_TIMER_CANCEL	request	Timer_name

11.5 Examples of Protocol Operation

To aid in understanding the flow of messages in the system, this section will be illustrated with a number of diagrams showing typical exchanges. These are to be interpreted only as examples of possible exchanges. In these diagrams, time progresses from top to bottom.

The box on the left, labeled APPLICATION ENTITY, is the originating application entity; to the left of this box are shown messages from and to the originating end-user process. The middle box on the page represents all of the intermediate layers from the presentation entity at the originator to the presentation entity at the target. The space between the two boxes represents the interface between the application and presentation layers, and diagonal lines are used to indicate

communication between the layers. On the right is another box representing the target application entity, with indication of messages to and from the target end-user process.

Horizontal lines cross the boxes, joining the lines which indicate messages entering and leaving each layer. In the application layer these horizontal lines represent boundaries between the states of the protocol. Each of these horizontal lines is numbered to correspond to a state transition described in section 13 of this report. The areas above and below each line contain the names of the current and resultant states, respectively.

In the box representing the presentation connection, the horizontal lines are labeled to indicate the type of application protocol data unit being transferred between the peer application entities. Parameters involved in the protocol data units or in the interface messages do not appear on these diagrams; the detailed specifications must be consulted for this information.

11.5.1 Connection Establishment

The normal procedure for the establishment of a connection between two peer application entities is illustrated in Figure 11-1. The end-user application process requests the establishment of an application connection. The application entity initiates a P_CONNECT request to the presentation entity. The peer presentation entity confirms the presentation connection, at which time the originating application uses the data transmission service of the presentation layer to send an ESTABLISH application protocol data unit to the target application entity. If the application connection is acceptable, the target user initiates an A_ESTABLISH response to the application layer, which constructs an A_ACCEPT application protocol data unit and transfers it to the presentation layer as part of a P_DATA request. This is returned to the originating application entity, and if conditions are acceptable the connection is confirmed to the originating end-user.

There are several stages of this exchange where establishment of the connection might be abandoned. If the target user does not accept the connection, it makes an A_ABORT request to the application layer, which transmits this information back to the originator. This is shown in Figure 11-2. The target application entity itself might find some unacceptable conditions, and reject the attempt to establish the connection, as illustrated in Figure 11-3.

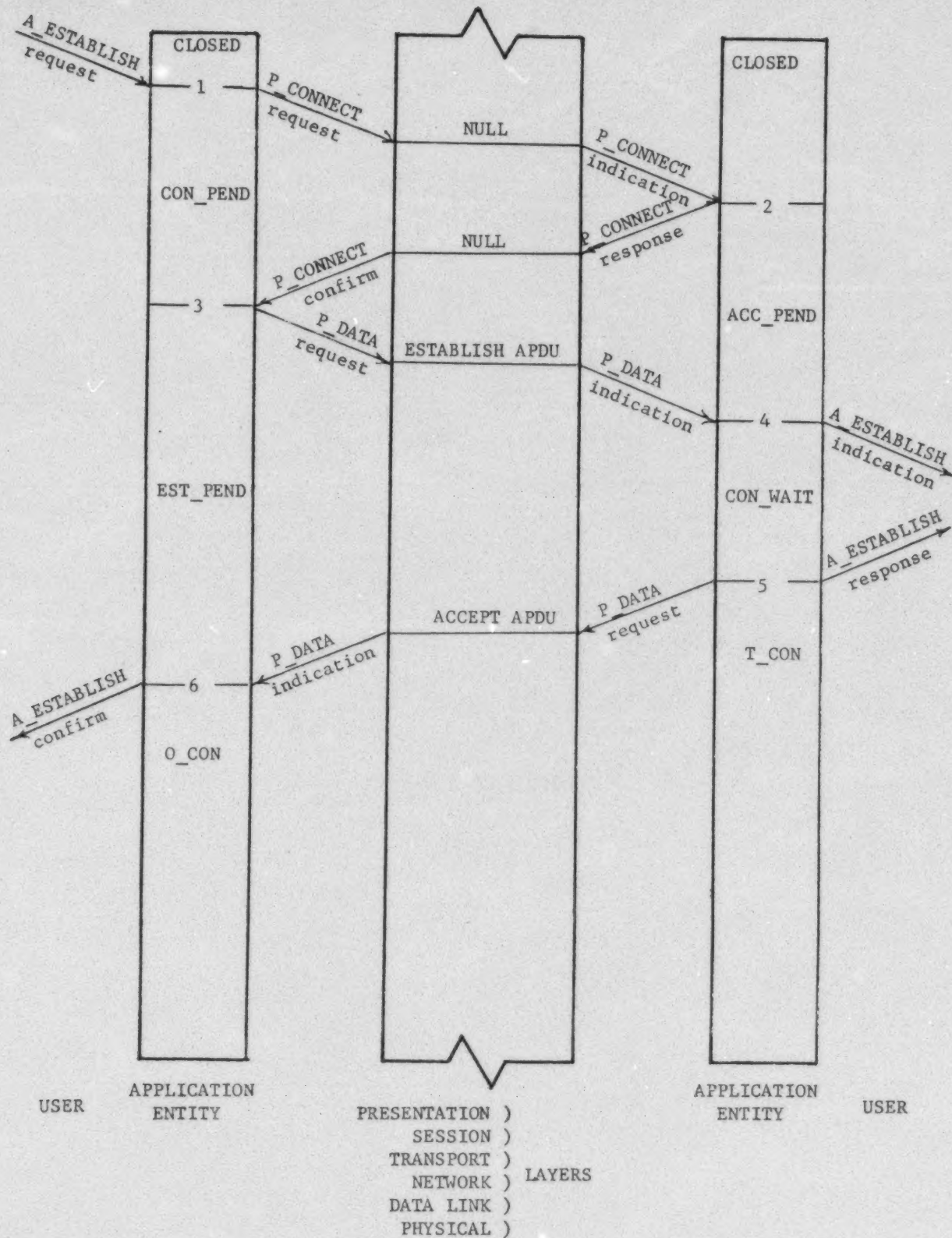


Figure 11-1 Normal establishment of a connection.

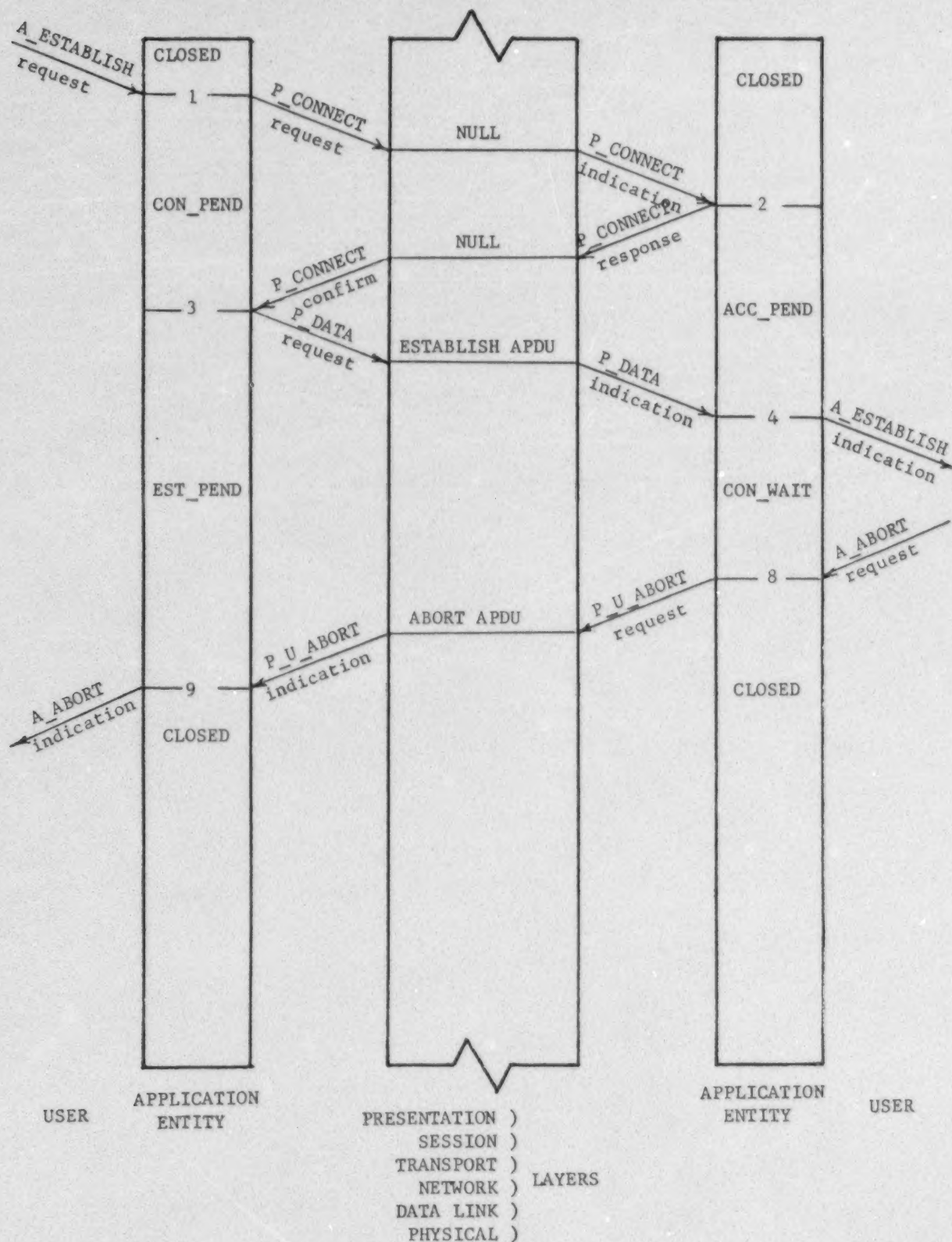


Figure 11-2 Target end-user does not accept connection.

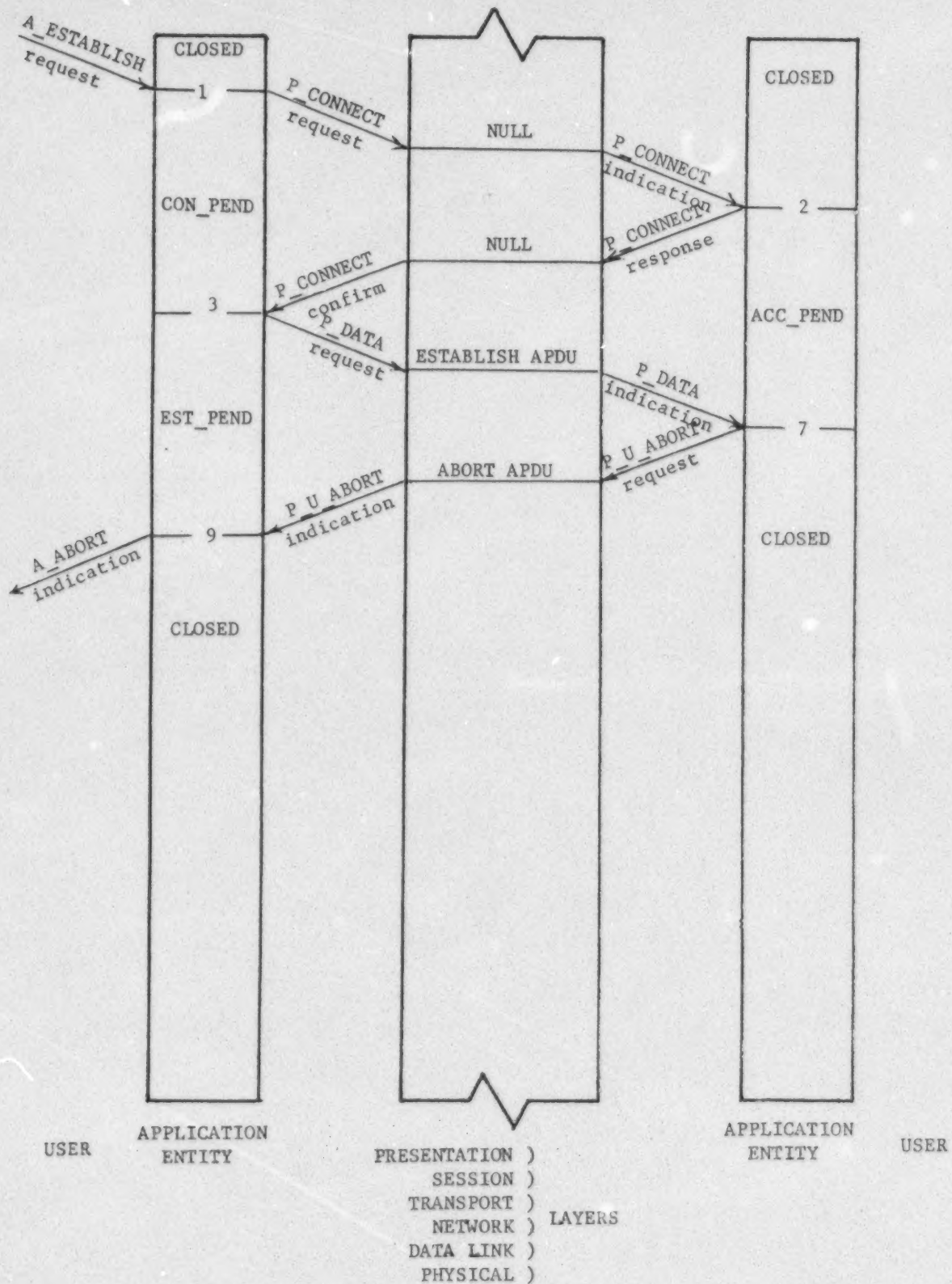


Figure 11-3 Connection not acceptable to target application entity.

Following the successful establishment of the application connection, both application entities are ready to transfer data within the selected subapplication.

11.5.2 Information Retrieval

Once a connection has been established with the phase set as information retrieval, the procedure for performing a search is illustrated in Figure 11-4. The originating end-user process requests a search, providing a search query as a parameter. The search query is transmitted to the target system by the use of a P_DATA.request of the presentation layer. At the target system the presentation layer initiates a P_DATA.indication to the application layer, which extracts the search query from the application protocol data unit and passes it to the target end-user process as a parameter of the A_SEARCH.indication.

The results of the search, possibly including one or more full or partial MARC records, are returned to the originating system by a similar process. This starts with an A_SEARCH.response from the target end-user process. The search results are encoded into a SEARCH_RESPONSE application protocol data unit, passed to the originating system, and returned to the originating end-user as parameters of an A_SEARCH.confirm event.

A similar procedure is followed to retrieve additional records from the result set of the search (A_PRESENT), or to obtain a single record specified by record key (A_FETCH), at any time.

After any of the information retrieval requests has been made by the originating end-user process, but before the corresponding confirmation event has occurred, the originating end-user process may initiate an A_CANCEL.request. This is transmitted to the target system by means of the presentation layer P_EXP_DATA.request. If the CANCEL APDU reaches the target application entity before it has received a response to the original request, an A_CANCEL.indication is passed to the target end-user process, which should then respond to the original request with a status code indicating the request was cancelled. This is illustrated in Figure 11-5.

If the target entity has already responded to the original request before receiving the cancel, it will ignore the cancel. The originating end-user process will receive the confirmation of the original request and will recognize from the status code that the cancel did not occur. This is illustrated in Figure 11-6.

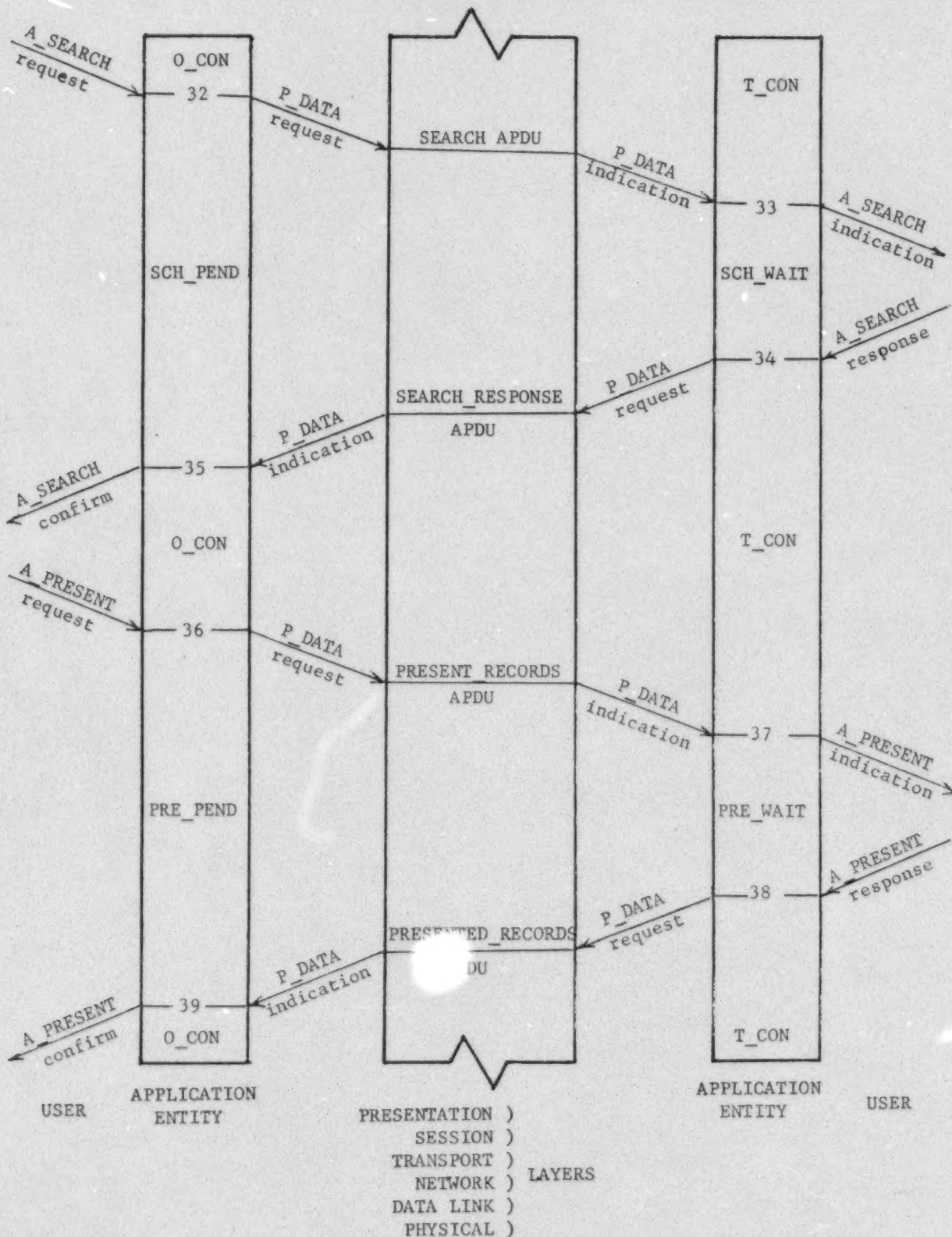


Figure 11-4 Normal search processing.

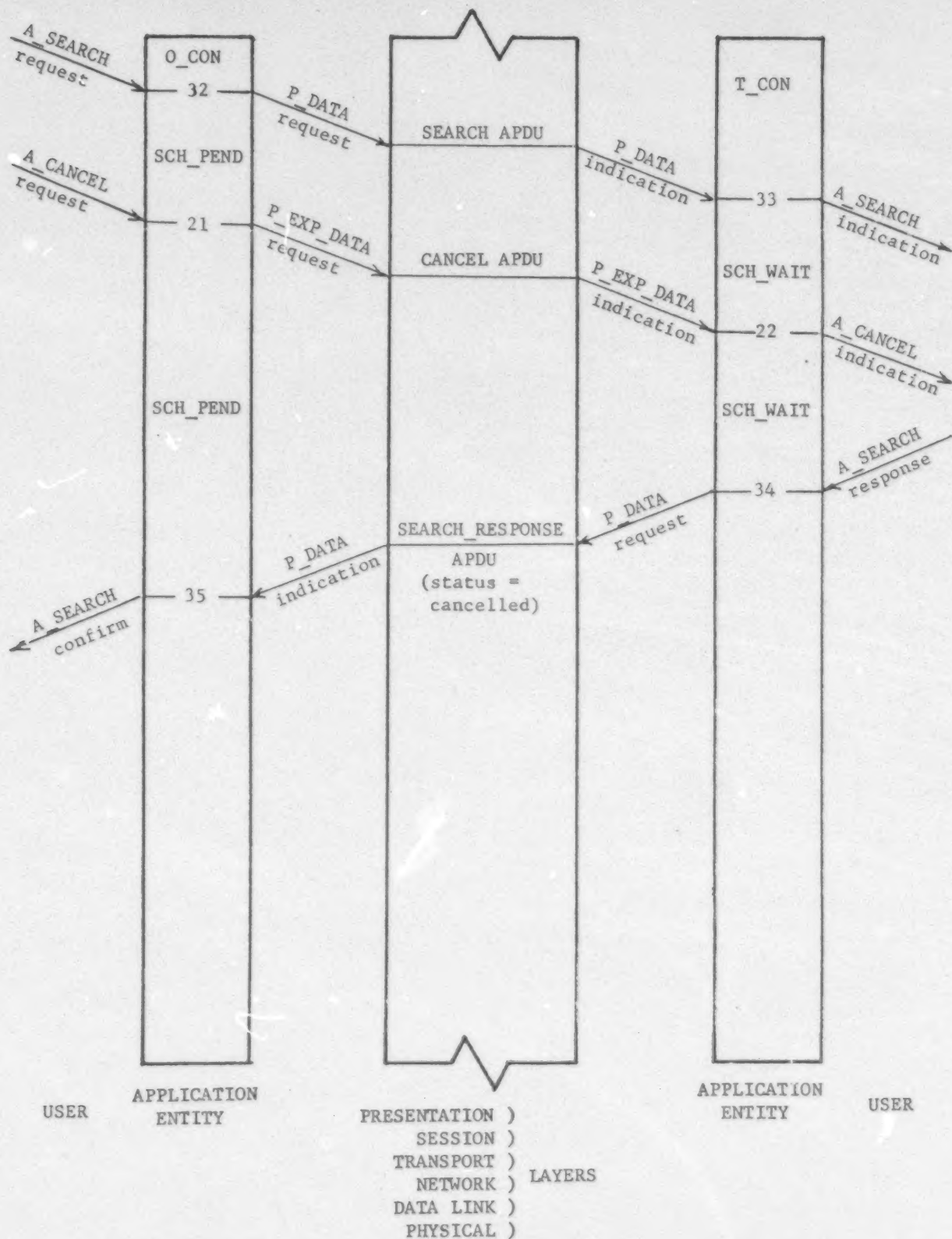


Figure 11-5 A request is cancelled.

11.5.3 Record Maintenance

Record maintenance involves two phases; the transmission of requests to the target system and a separate transmission of validation reports. For the reporting phase, the system which was the target during the original transmission of requests initiates another connection as the originating system for validation reports.

Three types of requests and corresponding validation reports are provided; add record, replace record, and delete record. The dialogue is very similar to that of the information retrieval phase, and so no additional figures are provided to illustrate the exchange.

The design of the protocol does not require that all record maintenance requests be validated. Whether or not validation is to be performed is indicated by one of the subapplication parameters which is set at the start of the connection (and which may be changed by the A_SET.request).

11.5.4 Mail Transmission

Mail transmission is very similar to record maintenance in that two phases are provided, one for transmission and one for reporting. The need for reporting can be indicated with each transmission, however, instead of being a subapplication parameter.

11.5.5 Connection Termination

Either origin or target may make an A_RELEASE.request to initiate an orderly termination of the connection. Figure 11-7 illustrates the exchange when the release is initiated by the origin system and Figure 11-8 illustrates the exchange when it is initiated by the target. The initiating system, when it receives the RELEASE_ANSWER APDU, confirms the release to the end-user and initiates a P_U_ABORT.request to the presentation layer to close the presentation connection.

When the release is initiated by the target system, it is possible that the release request will collide with another request from the origin. This situation is illustrated in Figure 11-9. The target system ignores the request and waits for a response to the release. A special case of this situation occurs when the request from the origin system is also a release. This allows the communication to be terminated with one fewer exchange of messages, as shown in Figure 11-10.

The connection may be abruptly terminated at any time by either end-user with the A_ABORT.request. This is illustrated in Figure 11-11.

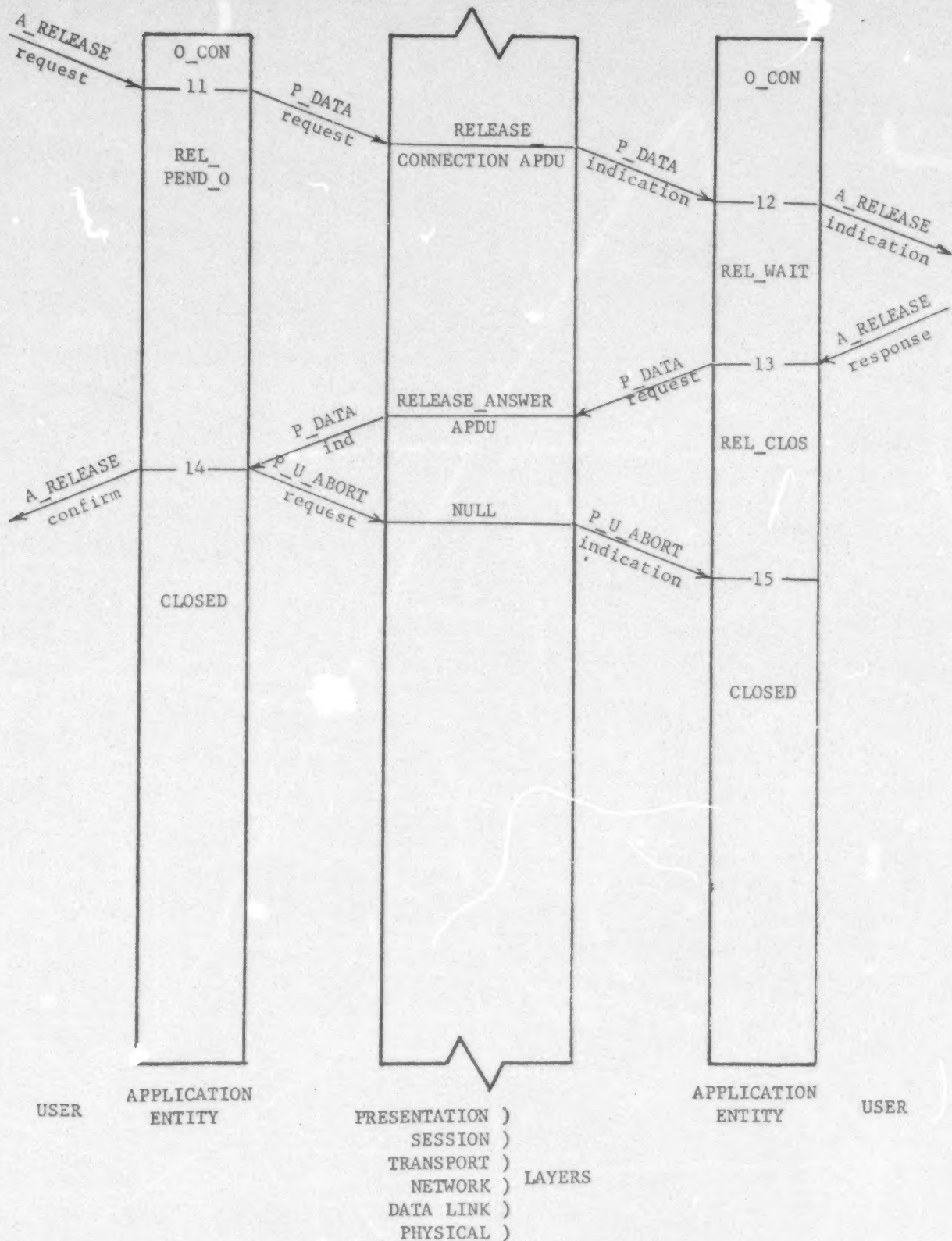


Figure 11-7 Release initiated by origin.

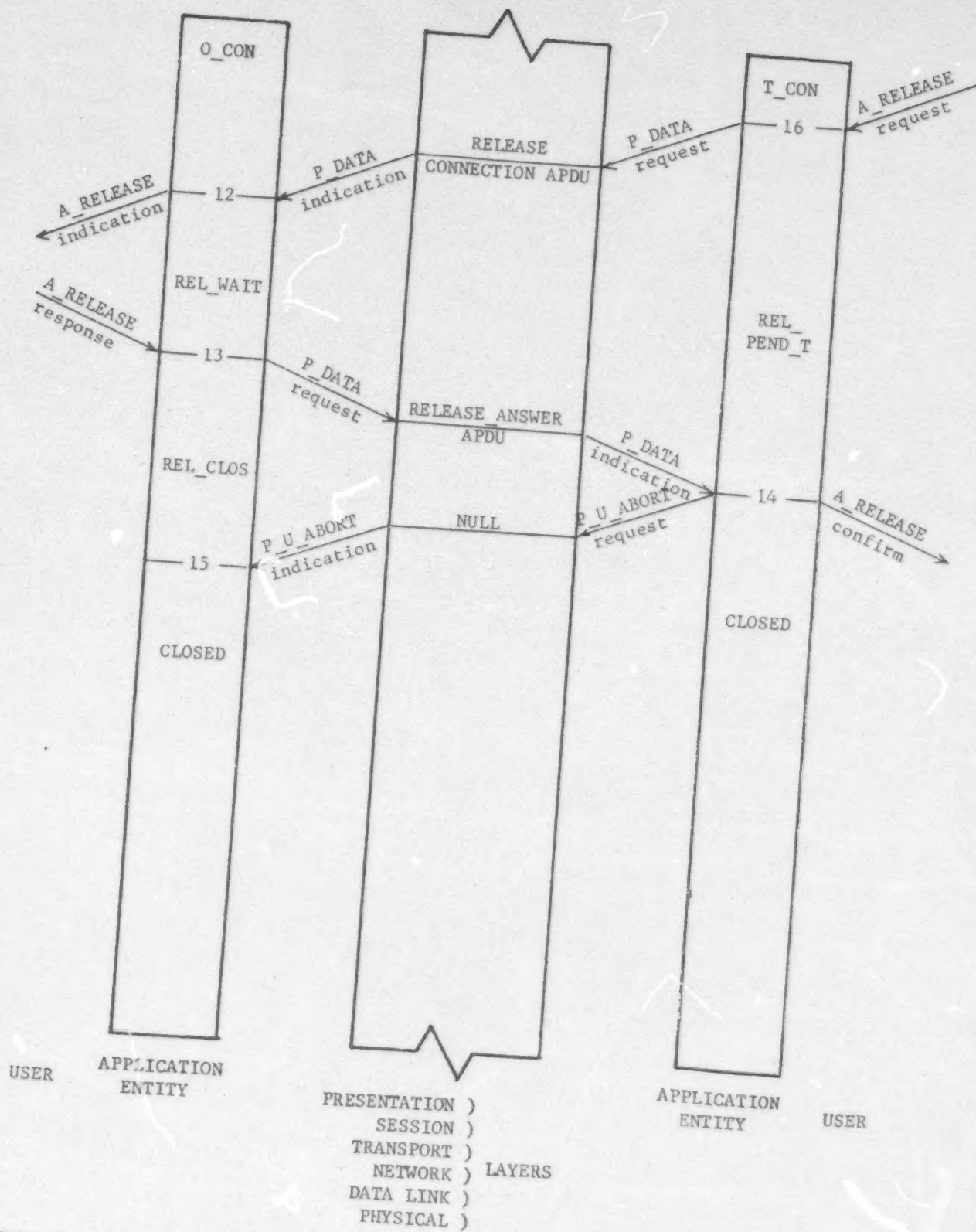


Figure 11-8 Release initiated by target.
 Application Level Protocol for Library Applications

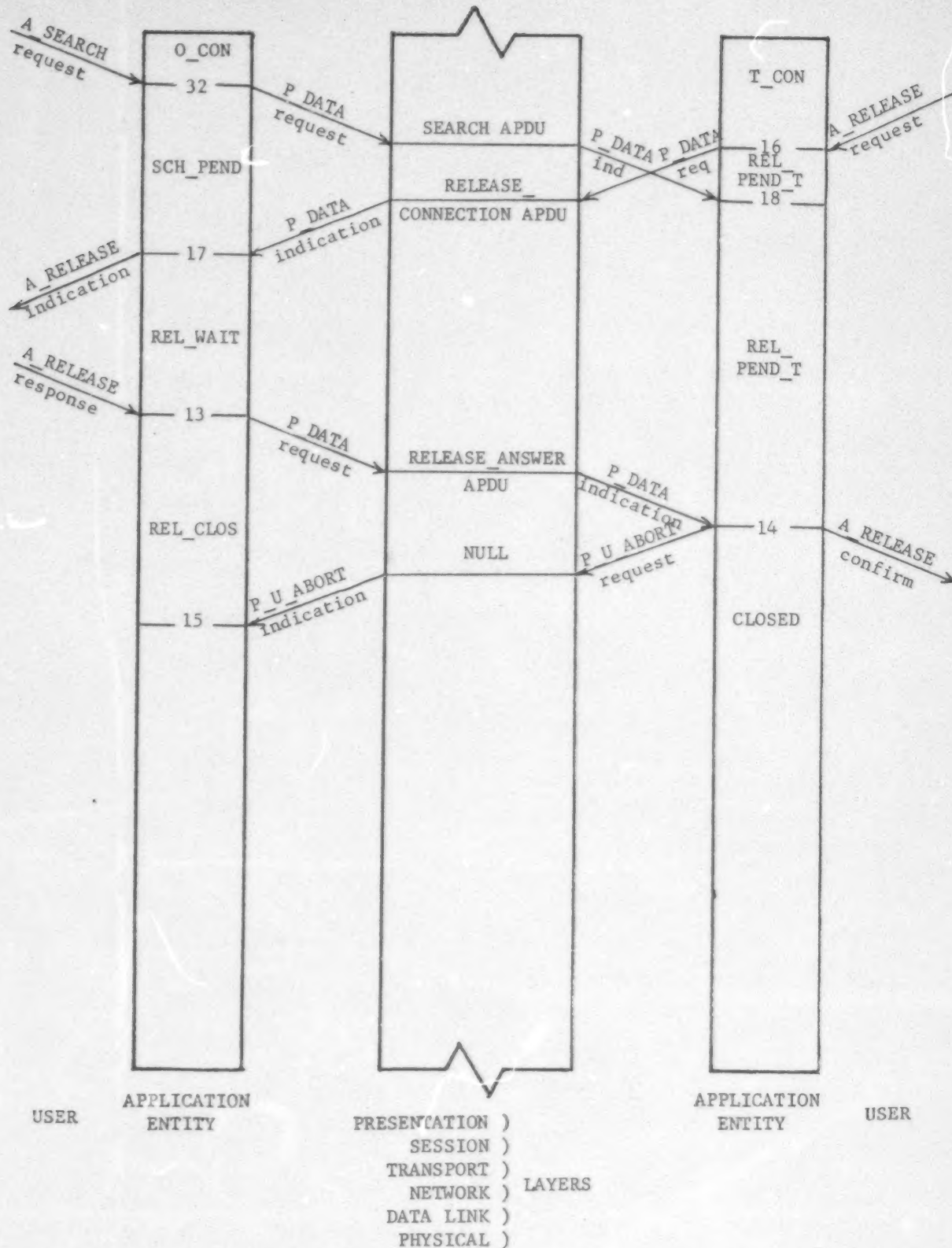


Figure 11-9 Release initiated by target, showing collision with request from origin.

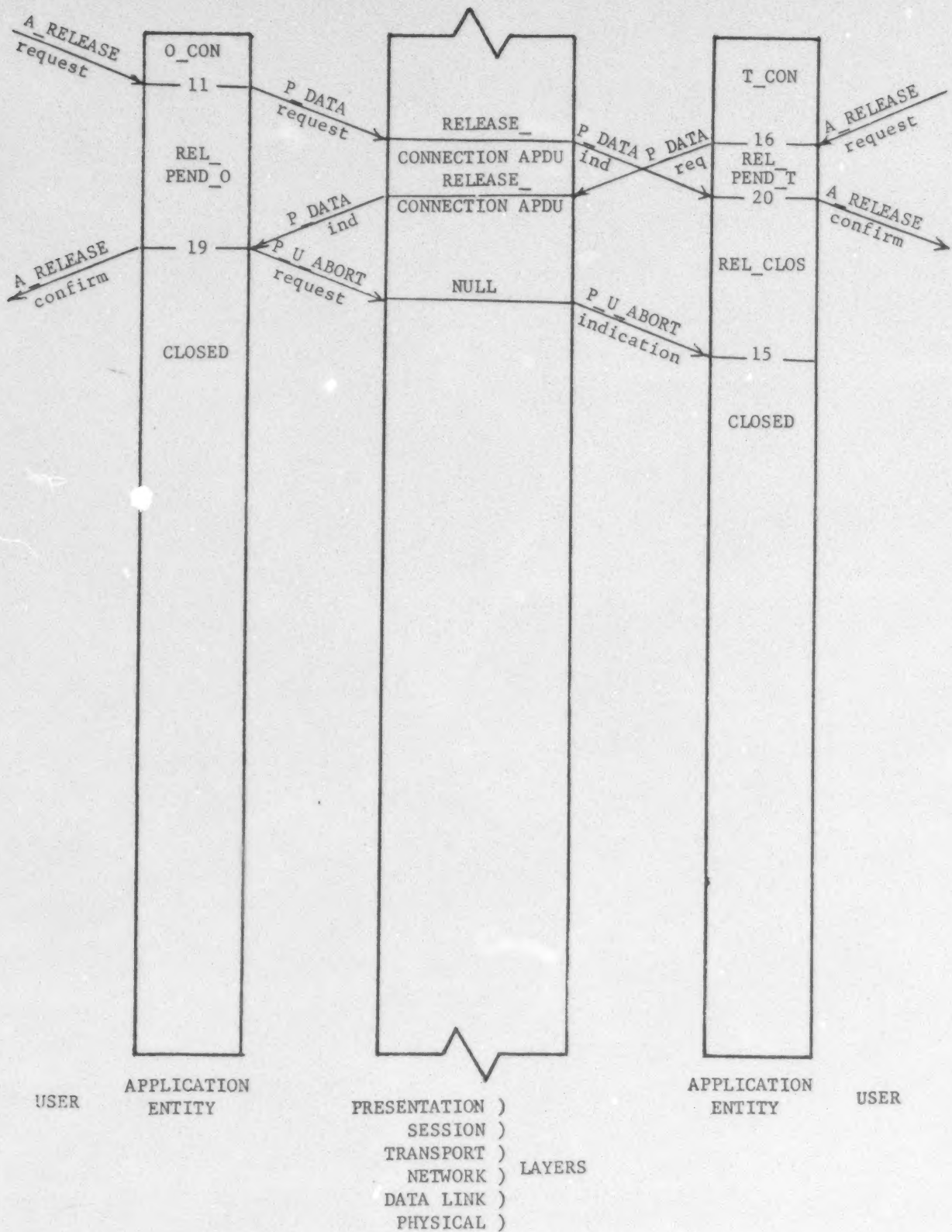


Figure 11-10 Release initiated by both origin and target.

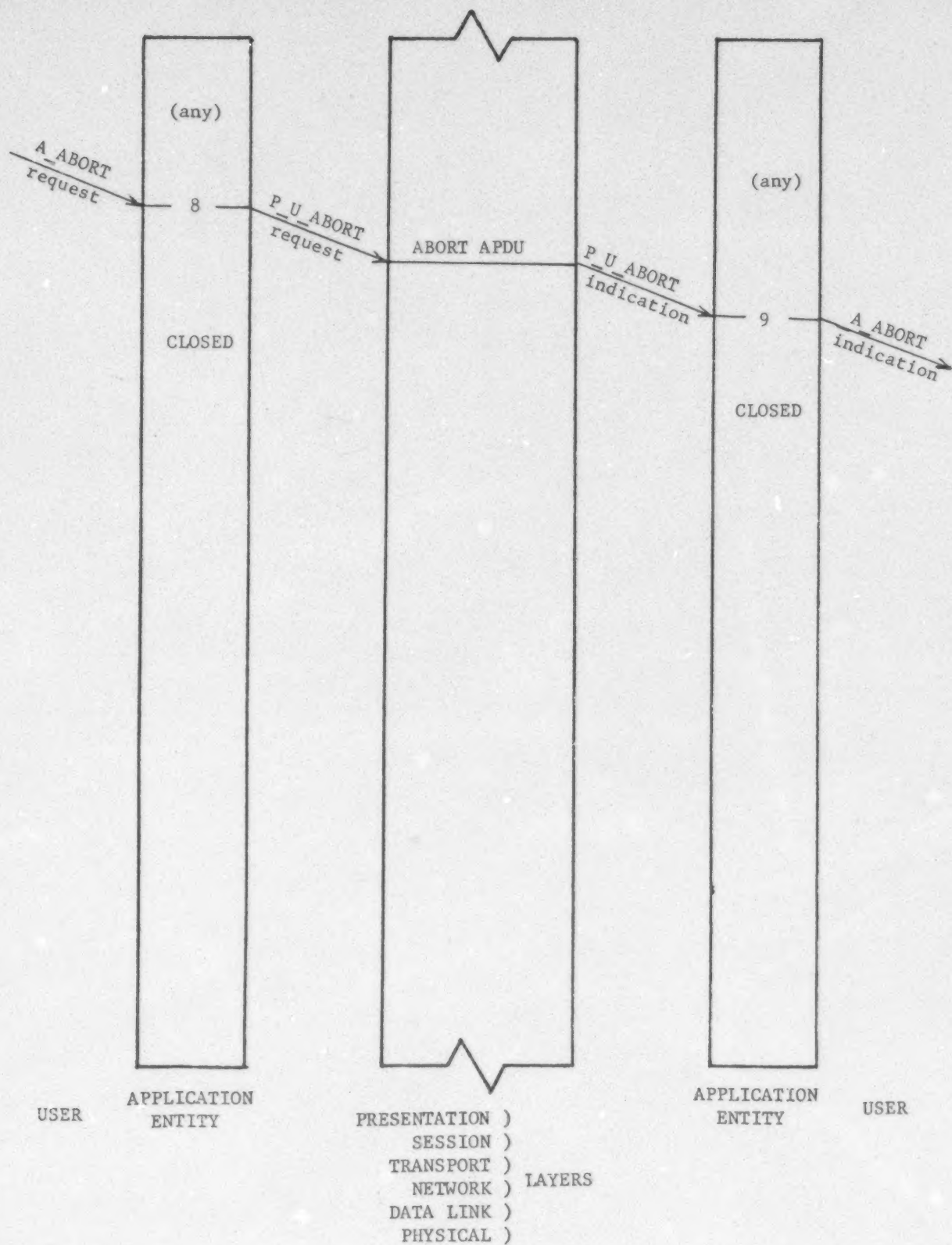


Figure 11-11 Abort initiated by origin end-user.

12. PROTOCOL SPECIFICATION METHOD AND NOTATION

12.1 General

The method of specifying the application protocol which is used in this report is adapted from that used in a number of reports prepared for the National Bureau of Standards [HLNP-81-02] [HLNP-81-04] [HLNP-81-05]. In these reports protocols are described by formally specifying "logical machines," or programs, performing the services of each layer in a system. Each of these machines has an interface with an entity in the layer above, an entity in the layer below, and to the operating system environment. In the case of the application layer, the "layer above" represents the end-user process. The description of the logical machine is abstract in that it is independent of any particular implementation. In a particular implementation, for example, multiple logical machines might be embodied in a single module. In the description which follows, the term "machine" is to be interpreted as "logical machine."

A machine communicates with its peer on another system by exchanging protocol data units with that peer. The format of these messages and the rules for their exchange constitute the definition of the protocol. These messages are sent by passing them to the next lower layer for transmission, and received by taking them from that lower layer. The rules for this exchange constitute the definition of the interface.

12.2 The Machine Model

In the NBS reports referenced above, a model employing a finite-state machine is used. Each input interface event initiates a series of program steps, resulting in one or more output interface events and a state transition. These steps are expressed in a subset of the PASCAL programming language, plus some extensions to reference interface events and their parameters. The notation used in this report has been simplified somewhat by the omission of details which do not contribute to understanding but only serve to make the statements formally correct.

An event occurs instantaneously as a result of an action by an entity in an adjacent layer or by the operating system environment, and is noticed by the entity to which it is directed at some unspecified later time. When an event is initiated, the initiating entity makes available information, i.e. outputs, in the form of interface parameters. Later, the entity to which the event is directed may "receive" this information as input when it recognizes the occurrence of the event. Note that initiating an event at an interface does not necessarily have an immediate

effect on the adjacent layer, and thus an entity requiring notice of the completion of an action performed by the adjacent layer must wait for a confirmation event to be initiated by that adjacent entity.

12.3 Description of the Specification Technique

The machine representing a protocol entity is described by a set of states and a set of transitions between them. Associated with each transition is a program segment. In addition the machine has designated initial and final states and specified initial values of the variables associated with the machine. These variables acquire the specified initial values whenever the machine begins its initial state. When it arrives at its final state, it logically ceases to exist.

The transitions of the machine are described by formulas, which typically have six parts:

- a. the transition number
- b. the resultant state
- c. the transition priority (optional)
- d. the current state
- e. the enabling condition (i.e., the interface event and optional predicates)
- f. the program segment

The enabling condition consists of an enabling interface event and optional "predicates" which must hold to enable the specified transition. Predicates may refer to the values of parameters associated with the interface event as well as to variables internal to the machine. Figure 12-1 illustrates a typical transition formula in schematic form.

```
#.  <RESULTANT_STATE> <---(transition priority) <CURRENT_STATE>
    [ INTERFACE EVENT ]
    ( PREDICATES )
```

PROGRAM SEGMENT

Figure 12-1. Schematic of a typical transition formula

Transitions are assigned numbers (#) as an aid in referencing them elsewhere in this report. As an example, consider the specification of the following transition:

```
1.  <STATE_2> <--- (4) <STATE_1>
    [ from A:EVENT.type ]
    ((P1 > e1) and
     (P2 = e2))
```

This formula states that the machine will make transition number 1 to state STATE_2 if it is in the state STATE_1 and if an interface event EVENT.type has occurred at the A-interface, making available parameters P1 and P2 (and possibly others which are not a part of the enabling condition). In addition P1 must exceed the value of the expression e1 and P2 must equal the value of the expression e2. This transition applies with priority 4; that is, the enabling conditions are tested in the order of their priorities; values of 1-3 have the higher priorities. When a transition is enabled, no further conditions are tested. Transitions with no specified priority are considered to be of the lowest priority.

Note that the interface event may have additional parameters associated with it, but their values are not part of the enabling condition. All event parameter values, however, become available when the event occurs and may be used in the program segment associated with the transition. Event-related parameters are referenced by an expression of the form

[from A:Parameter_name].

Since events occur instantaneously, the values associated with each parameter are those at the time of the event initiation.

Note also that this notation provides a label to identify the interface at which an event occurs. The labels used in specifying interfaces for the machine in this report are:

- U the interface between the application-user and the application-machine
- P the interface between the application-machine and the presentation-machine
- E the interface between the application-machine and the operating system environment

Associated with each transition is a program segment that describes the changes to variables internal to the machine and outputs made available at the various interfaces. The program segment associated with a transition occurs indivisibly, i.e. during the instant of the transition. Program segments may be null; in this case the effect of the transition is entirely embodied in the change of machine state.

12.4 Special Notational Devices

The model used in this report allows several devices to aid in the specification of the machine. They are:

- a. the abbreviation of multiple similar transitions
- b. the abbreviation of complex predicates
- c. the use of "primitives" (procedures and functions) to encapsulate implementation-dependent aspects of machine operation

12.4.1 Transition Abbreviation

The notation in this report provides for transition formulas in which the current state and the resultant state may, in fact, be labels defined to represent sets of states. Such state abbreviations are enumerated in section 13.2. For example if `waiting_state` is an abbreviation for `STATE1`, `STATE2`, and `STATE5`, it might appear in a transition formula such as the following:

```
<same_state> <--- <waiting_state>
  [ from A:EVENT ]
```

This formula is an abbreviation for the following fully specified set:

```
<STATE1> <--- <STATE1>
  [ from A:EVENT ]
<STATE2> <--- <STATE2>
  [ from A:EVENT ]
<STATE5> <--- <STATE5>
  [ from A:EVENT ]
```

Note the use of the set label <same_state> to designate that the new state is the same as the current state; this is the only state abbreviation which can be used as a resulting state, and it cannot be used as a current state. All other state abbreviations can be used only as a current state.

12.4.2 Predicate Abbreviation

The notation allows for the specific use of transition formulas in which the predicate element of the enabling condition may be a label defined to represent a list of conditions which must hold. Such abbreviation definitions have the following form:

```
predicate predicate_label;  
begin  
    predicate_expression  
end;
```

For example the following predicate abbreviation:

```
predicate Ack_ok;  
begin  
    First_sent <= Ack_number and  
    Ack_number < Next_to_send  
end;
```

might appear in a transition formula such as the following:

```
<STATE1> <--- <STATE2>  
    [from A:EVENT.type  
      (Ack_ok)]
```

This formula, then, is an abbreviation for the following fully specified transition:

```
<STATE1> <--- <STATE2>  
    [from A:EVENT.type  
      ((First_sent <= Ack_number) and  
       (Ack_number < Next_to_send))]
```

(This notation, however, has not been used in the present version of the protocol.)

12.4.3 Primitive Procedures

Procedures whose internal functioning is not an aspect of the protocol and is highly implementation-dependent may be specified. The use of these "primitive" procedures enhances the

abstract nature of protocol machine specifications by leaving unspecified such implementation-dependent details. Primitive procedures are specified by defining only their input parameters, resultant values if appropriate, and general action. They are defined in section 13.4.

12.4.4 Notation for Labels and Special Purposes

To aid in recognizing the various types of labels, the following usage of upper and lower case letters and special characters is used in this report:

<u>Usage</u>	<u>Description</u>	<u>Example</u>
UPPER	literals	TRANSFER
<UPPER>	state names	<CLOSED>
<lower>	state abbreviations	<active>
[dir A:UPPER.lower]	interface event names	[to U:A_CANCEL.confirm]
[dir A:Mixed]	interface event parameters	[to P:Operation]
lower	variables	appl_addr
lower()	functions and procedures	pdux(REASON)
lower(NULL)	functions with no parameters	pdux(NULL)

12.4.5 Presentation Layer Object List Notation

The presentation layer parameter Object_list is a structure consisting of a list of data types and data objects. The following notation is used when an object list is built:

```
[to P:Object_list := ((type, value),
                      (type, value), ...
                      (type, value))]
```

"type" is normally specified as a literal. "value" may be specified as an interface event parameter or as a primitive procedure. The following is an example of object list notation:


```

[ to P:Object_list :=
  ((HEADER, pduh( SEARCH_RESPONSE,
    [ from U:Search_status],
    [ from U:Result_count],
    [ from U:Result_set_ID],
    [ from U:Fullness_code],
    [ from U:Search_diagnostic],
    [ from U:Alternate_query],
    [ from U:Number_of_records])),
  (Z39_2, [ from U:Data]))]

```

When the object list is received from the presentation layer, the data type is not explicitly shown. Also, in place of the positional notation described above, each object in the list is referenced by name using Pascal record notation. For example, the object list above is referenced by

```

[ from P:Object_list.header]   and [ from P:Object_list.data]

```

13. FORMAL SPECIFICATION OF THE PROTOCOL

13.1 Interface Definitions

The definition of the interface between the application layer and the end-user process is given in Section 7 of Volume 1 of this report.

The definition of the interface between the application and presentation layers is given in Section 7 of the companion report Presentation Layer Protocol Development for Library and Information Science Applications. The definition of the Application Protocol Data Units, which are passed across that interface, is given in section 14.

13.2 State Descriptions

In this section each possible state of the protocol machine is listed alphabetically, with an indication of whether it can exist at the originating (O) or target (T) system, a brief description, and lists of state transitions which go from or to the state.

active (O,T)

This is the abbreviation representing all states except CLOSED.

From: 8, 10

active_1 (O,T)

This is an abbreviation representing all states except CLOSED and REL_CLOS.

From: 9

ACC_PEND (T)

A response has been made to a P_CONNECT indication received from the presentation layer.

From: 4, 7

To: 2

ADD_PEND (O)

An A_ADD_RECORD request has been transmitted to the target system.

From: 51

To: 48

ADD_WAIT (T)

An A_ADD_RECORD indication has been given to the end-user.

From: 50

To: 49

ADV_PEND (O)

An A_ADD_VALIDATED report has been transmitted to the target system.

From: 63

To: 60

ADV_WAIT (T)

An A_ADD_VALIDATED indication has been given to the end-user.

From: 62

To: 61

CLOSED (O,T)

This is the initial and final state of the logical machine.

From: 1, 2

To: 7, 8, 9, 10, 14, 15, 19

CON_PEND (O)

A P_CONNECT request has been made to the presentation layer.

From: 3

To: 1

CON_WAIT (O)

An A_ESTABLISH indication has been given to the end-user.

From: 5

To: 4

DEL_PEND (O)

An A_DELETE_RECORD request has been transmitted to the target system.

From: 59

To: 56

DEL_WAIT (T) An A_DELETE_RECORD indication has been given to the end-user.

From: 58

To: 57

DIS_PEND (O) An A_DISCARD_RESULT request has been transmitted to the target system.

From: 47

To: 44

DIS_WAIT (T)

An A_DISCARD_RESULT indication has been given to the end-user.

From: 46

To: 45

DLV_PEND

An A_DELETE_VALIDATED report has been transmitted to the target system.

From: 71

To: 68

DLV_WAIT (T)

An A_DELETE_VALIDATED indication has been given to the end-user.

From: 70

To: 69

EST_PEND (O)

An ESTABLISH APDU has been sent to the target system.

From: 6

To: 3

FCH_PEND (O)

An A_FETCH_RECORD request has been transmitted to the target system.

From: 43

To: 40

FCH_WAIT (T)

An A_FETCH_RECORD indication has been given to the end-user.

From: 42

To: 41

MRD_PEND (O)

An A_MAIL_READ report has been transmitted to the target system.

From: 79

To: 76

MRD_WAIT (T)

An A_MAIL_READ indication has been given to the end-user.

From: 78

To: 77

O_CON (O)

This is the connected "idle" state at the originating system.

From: 11, 24, 28, 32, 36, 40, 44, 48, 52, 56, 60, 64, 68, 72, 76

To: 6, 27, 31, 35, 39, 43, 47, 51, 55, 59, 63, 67, 71, 75, 79

PRE_PEND (O)

An A_PRESENT_RECORDS request has been transmitted to the target system.

From: 39

To: 36

PRE_WAIT (T)

An A_PRESENT_RECORDS indication has been given to the end-user.

From: 38

To: 37

REL_CLOS (O,T)

Connection termination is in progress, and the application entity is waiting for an abort indication from the presentation layer.

From: 15

To: 13, 20

rel_pend (O,T)

This is an abbreviation representing the states REL_PEND_O and REL_PEND_T.

From: 14

REL_PEND_O (O)

An A_RELEASE request has been transmitted to the target system.

From: 19

To: 11

REL_PEND_T (T)

An A_RELEASE request has been transmitted to the originating system.

From: 18, 20

To: 16, 18

REL_WAIT (T,O)

An A_RELEASE indication has been given to the end-user process.

From: 13

To: 12, 17

REP_PEND (O)

An A_REPLACE_RECORD request has been transmitted to the target system.

From: 55

To: 52

REP_WAIT (T)

An A_REPLACE_RECORD indication has been given to the end-user.

From: 54

To: 53

RPV_PEND (O)

An A_REPLACE_VALIDATED report has been transmitted to the target system.

From: 67

To: 64

RPV_WAIT (T)

An A_REPLACE_VALIDATED indication has been given to the end-user.

From: 66

To: 65

same_state (O,T)

To: 21, 22

SCH_PEND (O)

An A_SEARCH request has been sent to the target system.

From: 35

To: 32

SCH_WAIT (T)

An A_SEARCH indication has been given to the end-user.

From: 34

To: 33

SET_PEND (O)

An A_SET request has been sent to the target system.

From: 27

To: 24

SET_WAIT (T)

An A_SET indication has been given to the end-user.

From: 26

To: 25

SHW_PEND (O) An A_SHOW request has been sent to the target system.

From: 31

To: 28

SHW_WAIT (T)

An A_SHOW indication has been given to the end-user.

From: 30

To: 29

SML_PEND (O)

An A_SEND_MAIL request has been transmitted to the target system.

From: 75

To: 72

SML_WAIT (T)

An A_SEND_MAIL indication has been given to the end-user.

From: 74

To: 74

T_CON (T)

This is the connected "idle" state at the target system.

From: 16, 23, 25, 29, 33, 37, 41, 45, 49, 53, 57, 61, 65, 69, 73, 77

To: 5, 23, 26, 30, 34, 38, 42, 46, 50, 54, 58, 62, 66, 70, 74, 78

x_con (O,T)

This is an abbreviation for the following states:

O_CON, T_CON

From: 12

xxx_pend (O)

This is an abbreviation for the following states:

ADD_PEND, ADV_PEND, DEL_PEND, DIS_PEND, DLV_PEND, FCH_PEND, MRD_PEND, PRE_PEND, REP_PEND, RPV_PEND, SCH_PEND, SET_PEND, SHW_PEND, SML_PEND

From: 17, 21

xxx_wait (T)

This is an abbreviation for the following states:

ADD_WAIT, ADV_WAIT, DEL_WAIT, DIS_WAIT, DLV_WAIT, FCH_WAIT, MRD_WAIT, PRE_WAIT, REP_WAIT, RPV_WAIT, SCH_WAIT, SET_WAIT, SHW_WAIT, SML_WAIT

From: 22

13.3 Local Machine Variables

apdu_header: the header portion of an application protocol data unit.

conn_params: parameters which are specified by the originating end-user process at the start of a connection, and which may not be altered during the connection.

exp_pdu_type: the type of application protocol data unit expected by the application entity at the originating system in response to a request.

foreign_appl_addr: a code identifying the application at the peer system.

foreign_pres_addr: a code identifying the presentation address at the peer system.

local_appl_addr: a code identifying the application at the local system.

local_pres_addr: a code identifying the presentation address at the local system.

phase: a code indicating which of the several phases has been specified for operation of the system.

protocol_ID: a code identifying this protocol.

protocol_version: a code identifying the version of this protocol which is being used.

sub_appl_parms: parameters which apply to a particular subapplication and which may be changed by the originating system.

13.4 Primitive Procedures

acceptable_ESTAB(NULL)

a boolean function which determines if connection parameters are acceptable to the logical machine.

cnvrt_a_addr(application_address)

converts an application address to the corresponding presentation address.

cnvrt_p_addr(presentation_address)

converts a presentation address to the corresponding application address.

est_abort_reason(NULL)

supplies the reason code when connection parameters are not acceptable to the target application entity. Possible reasons:

- wrong protocol ID
- wrong protocol version

unrecognized subapplication ID

get_asap(NULL)

obtains the application service access point address for the current end-user application.

pduh(type,parameter1,parameter2, . . .)

creates an application protocol data unit header for the type of PDU specified.

pdux(field_name)

extracts the named field from the APDU header which is stored in the local variable apdu_header.

ptype(NULL)

supplies the type of an application protocol data unit which has been received from the presentation entity.

state_code(NULL)

supplies a code indicating the current phase of the application machine.

13.5 Initialization of the Machine

The following procedure is assumed to be executed before each transition from the CLOSED state of the application protocol machine.

```
begin
    conn_params := NULL;
    subappl_params := NULL;
    protocol_ID := LSP;
    protocol_version := 1;
end ;
```


13.6 State Transition Definitions

```
1.  <CON_PEND> <--- <CLOSED>
    [from U:A_ESTABLISH.request]

begin
  local_appl_addr := get_asap(NULL);
  foreign_appl_addr := [from U:Called_application];
  conn_parms := [from U:Connection_parameters];
  phase := [from U:Subapplication_ID];
  subappl_parms := [from U:Subapplication_parameters];
  local_pres_addr := cnvrt_a_addr(local_appl_addr);
  foreign_pres_addr := cnvrt_a_addr(foreign_appl_addr);
  [to P:P_CONNECT.request
    (Acceptor_address := foreign_pres_addr,
     Initiator_address := local_pres_addr,
     Initial_context_name := NULL,
     Initial_neg_right := NULL,
     Context_set := NULL,
     User_data := NULL)];
end ;
```

CURRENT STATE:

The application machine is closed.

ENABLING CONDITION:

An A_ESTABLISH.request event occurs at the user interface, indicating that the user desires establishment of an application connection.

ACTION:

The parameters accompanying the A_ESTABLISH request are obtained and saved. The primitive cnvrt_a_addr is used to convert the application addresses into presentation-user addresses. The application machine initiates a P_CONNECT.request event at the presentation layer interface.

RESULTANT STATE:

The application machine awaits a P_CONNECT.confirm event.

```

2.  <ACC_PEND> <--- <CLOSED>
    [from P:P_CONNECT.indication]

    begin
      local_pres_addr := [from P:Acceptor_address];
      foreign_pres_addr := [from P:Initiator_address];
      local_appl_addr := cnvrt_p_addr(local_pres_addr);
      foreign_appl_addr := cnvrt_p_addr(foreign_pres_addr);
      [to P:P_CONNECT.response];
    end ;

```

CURRENT STATE:

The application machine is closed.

ENABLING CONDITION:

A P_CONNECT.indication event occurs at the presentation layer interface.

ACTION:

The local and foreign application addresses are converted from the presentation addresses and saved. A P_CONNECT.response event is initiated at the presentation layer interface.

RESULTANT STATE:

The application machine awaits an ESTABLISH APDU from the originating system.

```

-----
3.  <EST_PEND> <--- <CON_PEND>
    [from P:P_CONNECT.confirm]

begin

    [to P:P_DATA.request
      (Operation := NULL,
       Object_list :=
         (HEADER, pduh(ESTABLISH,
                        protocol_ID,
                        protocol_version,
                        NULL,
                        phase,
                        conn_parms,
                        subappl_parms)))]];

end ;
-----

```

CURRENT_STATE:

The application machine is awaiting a P_CONNECT.confirm event.

ENABLING CONDITION:

A P_CONNECT.confirm event occurs at the presentation layer interface.

ACTION:

An ESTABLISH APDU is created and transmitted to the target system by means of a P_DATA.request event initiated at the presentation layer interface.

RESULTANT STATE:

The application machine awaits receipt of an ACCEPT APDU from the target system.

```

4.  <CON_WAIT> <--- <ACC_PEND>
    [from P:P_DATA.indication]
      ((ptype(NULL) = ESTABLISH) and
       acceptable_ESTAB(NULL))

    begin
      apdu_header := [from P:Object_list.header];
      conn_parms := pdux(CONN_PARMS;
      phase := pdux(SUBAPPLICATION_ID);
      subappl_parms := pdux(SUBAPPL_PARMS);
      [to U:A_ESTABLISH.indication
        (Called_application := local_appl_addr,
         Connection_parameters := conn_parms,
         Subapplication_ID := phase,
         Subapplication_parameters := subappl_parms)];
    end ;

```

CURRENT STATE:

The application machine is awaiting an ESTABLISH APDU from the originating system.

ENABLING CONDITION:

A P_DATA.indication event occurs at the presentation layer interface. An ESTABLISH APDU has been received and the associated parameters are acceptable to the local application machine.

ACTION:

The connection parameters, subapplication parameters, and phase are saved. An A_ESTABLISH.indication event is initiated at the interface to the end-user.

RESULTANT STATE:

The application machine awaits a response to the establish indication given to the end-user.

```

5. <T_CON> <--- <CON_WAIT>
   [from U:A_ESTABLISH.response]

   begin
     conn_params := [from U:Connection_parameters];
     subappl_params := [from U:Subapplication_parameters];
     [to P:P_DATA.request
      (Operation := NULL,
       Object_list :=
         (HEADER, pduh(ACCEPT,
                       protocol_version,
                       NULL,
                       conn_params,
                       subappl_params)))]
   end ;

```

CURRENT STATE:

The application machine is awaiting an ESTABLISH response from the end-user.

ENABLING CONDITION:

An A_ESTABLISH.response event occurs at the user interface.

ACTION:

An ACCEPT APDU is created and transmitted to the originating system by means of a P_DATA.request event initiated at the presentation layer interface. If any connection parameters or subapplication parameters are supplied by the end-user, they are included in the PDU.

RESULTANT STATE:

The application machine is in the target connected state awaiting a request from the originating system.

```

6. <O_CON> <--- <EST-PEND>
   [from P:P_DATA.indication]
     (from ptype(NULL) = ACCEPT)

   begin
     apdu_header := [from P:Object_list.header];
     if pdux(CONN_PARMS) <> NULL then
       conn_parms := pdux(CONN_PARMS);
     if pdux(SUBAPPL_PARMS) <> NULL then
       subappl_parms := pdux(SUBAPPL_PARMS);
     [to U:A_ESTABLISH.confirm
      (Connection_parameters := pdux(CONN_PARMS),
       Subapplication_parameters := pdux(SUBAPPL_PARMS))];
   end ;

```

CURRENT STATE:

The application machine is awaiting an ACCEPT APDU from the target system.

ENABLING CONDITION:

A P_DATA.indication event occurs at the presentation layer interface. An ACCEPT APDU has been received, indicating the willingness of the target application machine to establish an application connection.

ACTION:

If any connection parameters or subapplication parameters are contained in the PDU, they replace the parameters originally supplied by the originating end-user process. An A_ESTABLISH.confirm event is initiated at the user interface. Replacement parameters received from the target system are passed to the originating end-user process to determine if they are acceptable.

RESULTANT_STATE:

The application machine is in the origin connected state awaiting a request from the end-user.

```

7.  <CLOSED> <--- <ACC_PEND>
    [ from P:PDATA.indication
      ((ptype(NULL) = ESTABLISH) and not
       acceptable_ESTAB(NULL))

    begin
      [ to P:U_ABORT.request
        User_data := pduh(ABORT,
                          est_abort_reason(NULL),
                          state_code(NULL))];
    end ;

```

CURRENT STATE:

The application machine is awaiting an ESTABLISH APDU from the originating system.

ENABLING CONDITION:

A P_DATA.indication event occurs at the presentation layer interface. An ESTABLISH APDU has been received and the associated parameters are not acceptable to the local application machine.

ACTION:

A P_U_ABORT.request event is initiated at the presentation layer interface. The user data parameter carries an ABORT APDU containing the reason why the local application machine rejected the connection.

RESULTANT STATE:

The application machine is closed.

```

8.  <CLOSED> <--- <active>
    [from U:A_ABORT.request]

    begin
      [to P:P_U_ABORT.request
       (User_data := pduh(ABORT,
                           [from U:Abort_reason],
                           state_code(NULL)))]];
    end ;

```

CURRENT STATE:

The application machine is in any state except CLOSED.

ENABLING CONDITION:

An A_ABORT.request event occurs at the user interface. The end-user wishes to unilaterally terminate the connection.

ACTION:

A P_U_ABORT.request event is initiated at the presentation layer interface. The user data parameter carries an ABORT APDU containing the reason why the application user initiated the disconnect request.

RESULTANT STATE:

The application machine is closed.

```

9.  <CLOSED> <--- <active_1>
    [from P:P_U_ABORT.indication]
      (ptype(NULL) = ABORT)

    begin
      apdu_header := [from P:Object_list.header];
      [to U:A_ABORT.indication
        (Abort_reason := pdux(ABORT_REASON))];
    end ;

```

CURRENT STATE:

The application machine is in any state except CLOSED and REL_CLOS.

ENABLING CONDITION:

A P_U_ABORT.indication event occurs at the presentation layer interface. The connection was immediately terminated.

ACTION:

An A_ABORT.indication event is initiated at the user interface. The reason code contains the reason for abort supplied by the peer system.

RESULTANT STATE:

The application machine is closed.

```
10. <CLOSED> <-- <active>
    [ from P:P_P_ABORT.indication]

    begin
      [ to U:A_ABORT.indication
        (Abort_reason := [ from P:Reason])];
    end ;
```

CURRENT STATE:

The application machine is in any state except CLOSED.

ENABLING CONDITION:

A P_P_ABORT.indication event occurs are the presentation layer interface, indicating an abort caused by reasons internal to the service provider.

ACTION:

An A_ABORT.indication event is initiated at the user interface. The reason code indicates that the source of the abort was at the local system.

RESULTANT STATE:

The application machine is closed.

```

11. <REL_PEND_O> <--- <O_CON>
    [ from U:A_RELEASE.request ]

    begin
      [ to P:P_DATA.request
        (Operation := NULL,
          Object_list :=
            (HEADER, pduh(RELEASE_CONNECTION,
                          [ from U:Release_reason ],
                          [ from U:Statistics ])))];
    end ;

```

CURRENT STATE:

The application machine is in the origin connected state.

ENABLING CONDITION:

An A_RELEASE.request event occurs at the user interface.

ACTION:

An RELEASE_CONNECTION APDU is created and transmitted to the target system by means of a P_DATA.request event initiated at the presentation layer interface.

RESULTANT STATE:

The application machine awaits a response to the release request.

```

12. <REL_WAIT> <--- <x_con>
    [from P:P_DATA.indication]
      (PIYPE(NULL) = RELEASE_CONNECTION)

    begin
      apdu_header := [from P:Object_list.header];
      [to U:A_RELEASE.indication
        (Release_reason := pdux(RELEASE_REASON),
         Statistics := pdux(STATISTICS))];
    end ;

```

CURRENT STATE:

The application machine is in the connected state.

ENABLING CONDITION:

A P_DATA.indication event occurs at the presentation layer interface. A RELEASE_CONNECTION APDU has been received.

ACTION:

An A_RELEASE.indication event is initiated at the user interface. A release reason and optional statistics are passed as parameters to the end-user.

RESULTANT STATE:

The application machine awaits a response to the release indication given to the end-user.

```

13. <REL_CLOS> <-- <REL_WAIT>
    [from U:A_RELEASE.response]

    begin
      [to P:P_DATA.request
        (Operation := NULL,
         Object_list :=
           (HEADER, pduh(RELEASE_ANSWER,
                        [from U:Statistics])))];
    end ;

```

CURRENT STATE:

The application machine is awaiting a response to a release indication given to the end-user.

ENABLING CONDITION:

An A_RELEASE.response event occurs at the user interface.

ACTION:

A RELEASE_ANSWER APDU is created and transmitted to the peer system by means of a P_DATA.request event initiated at the presentation layer interface.

RESULTANT STATE:

The application machine awaits an abort indication from the presentation layer.

```

-----
14. <CLOSED> <-- <rel_pend>
    [from P:P_DATA.indication]
      (ptype(NULL) = RELEASE_ANSWER)

    begin
      apdu_header := [from P:Object_list.header];
      [to U:A_RELEASE.confirm
        (Statistics := pdux(STATISTICS))];
      [to P:P_U_ABORT.request
        (User_data := NULL)];
    end ;
-----

```

CURRENT STATE:

The application machine is awaiting the response to a release request from its peer system.

ENABLING CONDITION:

A P_DATA.indication event occurs at the presentation layer interface. A RELEASE_ANSWER APDU has been received.

ACTION:

An A_RELEASE.confirm event is initiated at the user interface, supplying optional statistics. A P_U_ABORT.indication event is initiated at the presentation layer interface.

RESULTANT STATE:

The application machine is closed.

15. <CLOSED> <--- <REL_CLOS>
[from P:P_U_ABORT.indication]

begin

end ;

CURRENT STATE:

The application machine has sent a RELEASE_ANSWER APDU to the peer system and is awaiting an ABORT indication.

ENABLING CONDITION:

A P_U_ABORT Indication event occurs at the presentation layer interface.

ACTION:

No action is taken.

RESULTANT STATE:

The application machine is closed.

```

16. <REL_PEND_T> <--- <T_CON>
    [ from U:A_RELEASE.request ]

    begin
      [ to P:P_DATA.request
        (Operation := NULL,
          Object_list :=
            (HEADER, pduh( RELEASE_CONNECTION,
                          [ from U:Release_reason ],
                          [ from U:Statistics ]))) ];
    end ;

```

CURRENT STATE:

The application machine is in the target connected state.

ENABLING CONDITION:

An A_RELEASE.request event occurs at the user interface. A release is being requested by the end-user at the target system.

ACTION:

A RELEASE APDU is created and transmitted to the originating system by means of a P_DATA.request event initiated at the presentation layer interface.

RESULTANT STATE:

The application machine awaits a response to the release request.

```

17. <REL_WAIT> <--- <xxx_pend>
    [from P:P_DATA.indication]
      (ptype(NULL) = RELEASE_CONNECTION);

begin
  apdu_header := [from P:Object_list.header];
  [to U:A_RELEASE.indication
    (Release_reason := pdux(RELEASE_REASON),
     Statistics := pdux(STATISTICS))];
end ;

```

CURRENT STATE:

The application machine at the originating system is awaiting the response to a request sent to the target system.

ENABLING CONDITION:

A P_DATA.indication event occurs at the presentation layer interface. A RELEASE_CONNECTION APDU has been received instead of a response to the request.

ACTION:

An A_RELEASE.indication event is initiated at the user interface.

RESULTANT STATE:

The application machine awaits a response to the release indication given to the end-user.

```
18. <REL_PEND_T> <--- <REL_PEND_T>
    [from P:P_DATA.indication]
    (ptype(NULL) <> RELEASE_CONNECTION)
```

```
begin
```

```
end ;
```

CURRENT STATE:

The application machine at the target system is awaiting a response to a release request sent to the originating system.

ENABLING CONDITION:

A P_DATA.indication event occurs at the presentation layer interface. An APDU other than RELEASE_CONNECTION has been received. A request from the originating system has crossed paths with the release request.

ACTION:

The request is ignored.

RESULTANT STATE:

The application machine continues to await a response to the release request.

```

19. <CLOSED> <--- <REL_PEND_C>
    [from P:P_DATA.indication]
      (ptype(NULL) = RELEASE_CONNECTION)

    begin
      apdu_header := [from P:Object_list.header];
      [to U:A_RELEASE.confirm
        (Statistics := pdux(STATISTICS))];
      [to P:P_U_ABORT.request
        (User_data := NULL)];
    end ;

```

CURRENT STATE:

The application machine at the originating system is awaiting the response to a release request sent to the target system.

ENABLING CONDITION:

A P_DATA.indication event occurs at the presentation layer interface. A RELEASE_CONNECTION APDU has been received, indicating that the target system is also requesting a release.

ACTION:

An A_RELEASE.confirm event is initiated at the user interface, treating the release connection indication as if it were a response to the release request.

RESULTING STATE:

The application machine is closed.

```

20. <REL_CLCS> <--- <REL_PEND_T>
    [from P:P_DATA.indication]
    (ptype(NULL) = RELEASE_CONNECTION)

begin
    apdu_header := [from P:Object_list.header];
    [to U:A_RELEASE.confirm
    (Statistics := pdux(STATISTICS))];
end ;

```

CURRENT STATE:

The application machine at the target system is awaiting a response to a release request sent to the originating system.

ENABLING CONDITION:

A P_DATA.indication event occurs at the presentation layer interface. A RELEASE_CONNECTION APDU has been received, indicating that the originating system is also requesting a release.

ACTION:

An A_RELEASE.confirm event is initiated at the user interface, treating the release connection indication as if it were a response to the release request.

RESULTANT STATE:

The application machine awaits an abort indication from the presentation layer.

```

21. <same_state> <--- <xxx_pent>
    [from U:A_CANCEL.request]

    begin
      [to P:P_EXP_DATA.request
        (Operation := NULL,
         Object_list :=
           (HEADER, pduh(CANCEL,
                        [from U:Cancel_reason])));
    end ;

```

CURRENT STATE:

The application machine at the originating system is waiting for a response to a request sent to the target system.

ENABLING CONDITION:

An A_CANCEL.request event occurs at the user interface, indicating that the end-user wishes to cause processing of the request to stop.

ACTION:

A CANCEL APDU is created and transmitted to the target system by means of a P_EXP_DATA.request event initiated at the presentation layer interface.

RESULTANT STATE:

The application machine awaits either a response to its original request.

```

22. <same_state> <--- <xxx_wait>
    [ from P:P_EXP_DATA.indication ]
      (ptype(NULL) = CANCEL)

begin
  apdu_header := [ from P:Object_list.header];
  [ to U:A_CANCEL.indication
    (Cancel_reason := pdux(CANCEL_REASON))];
end ;

```

CURRENT STATE:

The application machine is waiting for a response to an indication given to the target end-user.

ENABLING CONDITION:

A P_EXP_DATA.indication event occurs at the presentation layer interface. A CANCEL APDU has been received.

ACTION:

An A_CANCEL.indication event is initiated at the user interface.

RESULTANT STATE:

The application machine awaits a response to the original indication given to the end-user.

```
23. <T_CON> <--- <T_CON>
    [ from P:P_EXP_DATA.indication ]
      (ptype(NULL) = CANCEL)

    begin

    end ;
```

CURRENT STATE:

The application machine is in the target connected state.

ENABLING CONDITION:

A P_EXP_DATA.indication event occurs at the presentation layer interface. A CANCEL APDU has been received. The target system has already responded to the request which is being cancelled.

ACTION:

The cancel indication is ignored.

RESULTANT STATE:

The application machine remains in the target connected state.

```

-----
24. <SET_PEND> <--- <O_CON>
    [from U:A_SET.request]

    begin
      [to P:P_DATA.request
        (Operation := NULL,
         Object_list :=
           (HEADER, pduh(SET,
                        [from U:Subapplication_parameters]))];
      exp_pdu_type := SET_RESPONSE;
    end ;
-----

```

CURRENT STATE:

The application machine is in the origin connected state.

ENABLING CONDITION:

An A_SET.request event occurs at the user interface.

ACTION:

A SET APDU is created and transmitted to the target system by means of a P_DATA.request event initiated at the presentation layer interface.

RESULTANT STATE:

The application machine awaits a response to the set request.

```

25. <SET_WAIT> <--- <T_CON>
    [ from P:P_DATA.indication ]
      (ptype(NULL) = SET)

    begin
      apdu_header := [ from P:Object_list.header ];
      subappl_parms := pdux(SUBAPPLICATION_PARMS)];
      [ to U:A_SET.indication
        (Subapplication_parameters := subappl_parms)];
    end ;

```

CURRENT STATE:

The application machine is in the target connected state.

ENABLING CONDITION:

A P_DATA.indication event occurs at the presentation layer interface. A SET APDU has been received.

ACTION:

An A_SET.indication event is initiated at the user interface. The subapplication parameters are saved and also passed to the end-user.

RESULTANT STATE:

The application machine awaits a response to the set indication given to the end-user.

```

26. <T_CON> <--- <SET_WAIT>
    [from U:A_SET.response]

    begin
        subappl_parms := [from U:Subapplication_parameters];
        [to P:P_DATA.request
        (Operation := NULL,
        Object_list :=
            (HEADER, pduh(SET_RESPONSE,
                [from U:Set_status],
                subappl_parms)))]];
    end ;

```

CURRENT STATE:

The application machine is awaiting a response to a set indication given to the end-user.

ENABLING CONDITION:

An A_SET.response event occurs at the user interface.

ACTION:

A SET_RESPONSE APDU is created and transmitted to the originating system by means of a P_DATA.request event initiated at the presentation layer interface.

RESULTANT STATE:

The application machine returns to the target connected state to await another request from the originating system.

```

-----
27. <O_CON> <--- <SET_PEND>
    [from U:P_DATA.indication]
      (ptype(NULL) = SET_RESPONSE)

begin
  apdu_header := [from P:Object_list.header];
  if pdux(SUBAPPL_PARMS) <> NULL then
    subappl_parms := pdux(SUBAPPL_PARMS);
  [to U:A_SET.confirm
    (Subapplication_parameters := pdux(SUBAPPL_PARMS),
     Set_status := pdux(SET_STATUS))];
end ;
-----

```

CURRENT STATE:

The application machine is awaiting a response to a set request sent to the target system.

ENABLING CONDITION:

A P_DATA.indication event occurs at the presentation layer interface. A SET_RESPONSE APDU has been received.

ACTION:

An A_SET.confirm event is initiated at the user interface. Any subapplication parameters which are received are saved and also passed as parameters to the end-user.

RESULTANT STATE:

The application machine returns to the origin connected state to await another request from the end-user.

```

29. <SHW_PEND> <--- <O_CON>
    [ from U:A_SHOW.request ]

    begin
      [ to P:P_DATA.request
        (Operation := NULL,
          Object_list :=
            (HEADER, pduh( SHOW,
                          [ from U:Service_options ])) )];
      exp_pdu_type := SHOW_RESPONSE;
    end ;

```

CURRENT STATE:

The application machine is in the origin connected state.

ENABLING CONDITION:

An A_SHOW.request event occurs at the user interface.

ACTION:

A SHOW APDU is created and transmitted to the target system by means of a P_DATA.request event initiated at the presentation layer interface.

RESULTANT STATE:

The application machine awaits a response to the show request.

```

29. <SHW_WAIT> <--- <T_CON>
    [from P:P_DATA.indication]
      (ptype(NULL) = SHOW)

    begin
      apdu_header := [from P:Object_list.header];
      [to U:A_SHOW.indication
        (Service_options := pdux(SERVICE_OPTIONS))];
    end ;

```

CURRENT STATE:

The application machine is in the target connected state.

ENABLING CONDITION:

A P_DATA.indication event occurs at the presentation layer interface. A SHOW APDU has been received.

ACTION:

An A_SHOW.indication event is initiated at the user interface. The service options are passed to the end-user as parameters.

RESULTANT_STATE:

The application machine awaits a response to the show indication given to the end-user.

```

30. <T_CON> <--- <SHW_WAIT>
    [from U:A_SHOW.response]

    begin
      [ to P:P_DATA.request
        {Operation := NULL,
         Object_list :=
           (HEADER, pduh(SHOW_RESPONSE,
                        service_options)))];
    end ;

```

CURRENT STATE:

The application machine is awaiting a response to a show indication given to the end user.

ENABLING CCNDITION:

A A_SHOW.response event occurs at the user interface.

ACTION:

A SHOW_RESPONSE APDU is created and transmitted to the originating system by means of a P_DATA.request event initiated at the presentation layer interface.

RESULTANT STATE:

The application machine returns to the target connected state.

```

31. <O_CON> <--- <SHW_PEND>
    [from U:P_DATA.indication]
      (ptype(NULL) = SHOW_RESPONSE)
    begin
tb 18 apdu_header := [from P:Object_list.header];
    [to U:A_SHOW.confirm
      Service_options := pdux(SERVICE_OPTIONS)];
    end ;

```

CURRENT STATE:

The application machine is waiting for a response to a show request sent to the target system.

ENABLING CONDITION:

A P_DATA.indication event occurs at the presentation layer interface. A SHOW_RESPONSE APDU has been received.

ACTION:

An A_SHOW.confirm event is initiated at the end-user interface.

RESULTANT STATE:

The application machine returns to the origin connected state to await another request from the end-user.

```

32. <SCH_PEND> <--- <O_CON>
    [from U:A_SEARCH.request]
      (phase = INFORMATION_RETRIEVAL)

begin
  [to P:P_DATA.request
    (Operation := NULL,
     Object_list :=
       (HEADER, pduh(SEARCH,
                     [from U:Query])));
   exp_pdu_type := SEARCH_RESPONSE;
end ;

```

CURRENT STATE:

The application machine is in the origin connected state.

ENABLING CONDITION:

An A_SEARCH.request event occurs at the user interface and the phase is information retrieval.

ACTION:

A SEARCH APDU is created and transmitted to the target system by means of a P_DATA.request event initiated at the presentation layer interface.

RESULTANT STATE:

The application machine awaits a response to the search request.

```

33. <SCH_WAIT> <--- <T_CON>
    [ from P:P_DATA.indication
      ((ptype(NULL) = SEARCH) and
        (phase = INFORMATION_RETRIEVAL))

    begin
      apdu_header := [ from P:Object_list.header ];
      [ to U:A_SEARCH.indication
        (Query := pdux(QUERY)) ];
    end ;

```

CURRENT STATE:

The application machine is in the target connected state.

ENABLING CONDITION:

A P_DATA.indication event occurs at the presentation layer interface. A SEARCH APDU has been received and the phase is information retrieval.

ACTION:

An A_SEARCH.indication event is initiated at the user interface. The search query is passed as a parameter to the end-user.

RESULTANT STATE:

The application machine awaits a response to the search indication given to the end-user.

```

34. <T_CON> <--- <SCH_WAIT>
    [from U:A_SEARCH.response]

begin
  [to P:P_DATA.request
    (Operation := NULL,
     Object_list :=
       ((HEADER, pduh( SEARCH_RESPONSE,
                        [ from U:Search_status],
                        [ from U:Result_count],
                        [ from U:Fullness_code],
                        [ from U:Result_set_ID],
                        [ from U:Search_diagnostic],
                        [ from U:Alternate_query],
                        [ from U:Number_of_records])),
       (Z39_2, [from U:Data]))]);
end ;

```

CURRENT STATE:

The application machine is awaiting a response to a search indication given to the end-user.

ENABLING CONDITION:

An A_SEARCH.response event occurs at the user interface.

ACTION:

A SEARCH_RESPONSE APDU is created and transmitted to the originating system by means of a P_DATA.request event initiated at the presentation layer interface.

RESULTANT STATE:

The application machine returns to the target connected state to await another request from the originating system.

```

35. <O_CON> <--- <SCH_PEND>
    [from U:P_DATA.indication]
      (ptype(NULL) = SEARCH_RESPONSE)

begin
  apdu_header := [from P:Object_list.header];
  [to U:A_SEARCH.confirm
    (Search_status := pdux(SEARCH_STATUS),
     Result_count := pdux(RESULT_COUNT),
     Result_set_ID := pdux(RESULT_SET_ID),
     Fullness_code := pdux(FULLNESS_CODE),
     Search_diagnostic := pdux(SEARCH_DIAGNOSTIC),
     Alternate_query := pdux(ALTERNATE_QUERY),
     Number_of_records := pdux(NUMBER_OF_RECORDS),
     Data := [from P:Object_list.data]);];
end ;

```

CURRENT STATE:

The application machine is awaiting the response to a search request sent to the target system.

ENABLING CONDITION:

A P_DATA.indication event occurs at the presentation layer interface. A SEARCH_RESPONSE APDU has been received.

ACTION:

An A_SEARCH.confirm event is initiated at the end-user interface, supplying the results of the search.

RESULTANT STATE:

The application machine returns to the origin connected state to await another request from the end-user.

```

36. <PRE_PEND> <-- <O_CON>
    [from U:A_PRESENT.request]
      (phase = INFORMATION_RETRIEVAL)

begin
  [to P:P_DATA.request
    (Operation := NULL,
     Object_list :=
       (HEADER, pduh(PRESENT_RECORDS,
                     [from U:Fullness_code],
                     [from U:Result_set_position],
                     [from U:Number_of_records],
                     [from U:Result_set_ID])));
    exp_pdu_type := PRESENTED_RECORDS;
  end ;

```

CURRENT STATE:

The application machine is in the origin connected state.

ENABLING CONDITION:

An A_PRESENT RECORDS.request event occurs at the user interface and the phase is information retrieval.

ACTION:

A PRESENT_RECORDS APDU is created and transmitted to the target system by means of a P_DATA.request event at the presentation layer interface.

RESULTANT STATE:

The application machine awaits a response to the present records request.

```

37. <PRE_WAIT> <--- <T_CON>
    [from P:PDATA.indication]
      ((ptype(NULL) = PRESENT_RECORDS) and
       (phase = INFORMATION_RETRIEVAL))

begin
  apdu_header := [from P:Object_list.header];
  [to U:A_PRESENT.indication
    (Fullness_code := pdux(FULLNESS_CODE),
     Result_set_position := pdux(RESULT_SET_POSITION),
     Number_of_records := pdux(NUMBER_OF_RECORDS),
     Result_set_ID := pdux(RESULT_SET_ID))];
end ;

```

CURRENT STATE:

The application machine is in the target connected state.

ENABLING CONDITION:

A P_DATA.indication event occurs at the presentation layer interface. A PRESENT_RECORDS APDU has been received and the phase is information retrieval.

ACTION:

An A_PRESENT.indication event is initiated at the user interface. Parameters specifying the records to be supplied are passed to the end-user.

RESULTANT STATE:

The application machine awaits a response to the present records indication given to the end-user.

```

-----
38. <T_CON> <-- <PRE_WAIT>
    [from U:A_PRESENT.response]

    begin
      [to P:P_DATA.request
        (Operation := NULL,
         Object_list :=
           (HEADER, pduh(PRESENTED_RECORDS,
                        [from U:Present_status],
                        [from U:Result_set_position],
                        [from U:Number_of_records])),
         (Z39_2, [from U:Data]))];
    end ;
-----

```

CURRENT STATE:

The application machine is awaiting a response to a present records indication given to the end-user.

ENABLING CONDITION:

An A_PRESENT.response event occurs at the user interface.

ACTION:

A PRESENTED_RECORDS APDU is created and transmitted to the originating system by means of a P_DATA.request event initiated at the presentation layer interface.

RESULTANT STATE:

The application machine returns to the target connected state to await another request from the originating system.

```

-----
39. <Q_CON> <--- <PRE_PEND>
    [ from U:P_DATA.indication ]
      (ptype(NULL) = PRESENTED_RECORDS)

    begin
      apdu_header := [ from P:Object_list.header ];
      [ to U:A_PRESENT.confirm
        ( Present_status := pdux(PRESENT_STATUS),
          Result_set_position := pdux(RESULT_SET_POSITION),
          Number_of_records := pdux(NUMBER_OF_RECORDS),
          Data := [ from P:Object_list.data ] ) ];
    end ;
-----

```

CURRENT STATE:

The application machine is awaiting the response to a present records request sent to the target system.

ENABLING CONDITION:

A P_DATA.indication event occurs at the presentation layer interface. A PRESENTED_RECORDS APDU has been received.

ACTION:

An A_PRESENT.confirm event is initiated at the end-user interface, supplying the requested records.

RESULTANT STATE:

The application machine returns to the origin connected state to await another request from the end-user.


```

-----
40. <FCH_PEND> <--- <O_CON>
    [ from U:A_FETCH_RECORD.request ]
      ( phase = INFORMATION_RETRIEVAL )

    begin
      [ to P:P_DATA.request
        ( Operation := NULL,
          Object_list :=
            ( HEADER, pduh( FETCH_RECORD,
                           [ from U:Fullness_code ],
                           [ from U:Record_key ],
                           [ from U:File_name ] ) ) ) ];
      exp_pdu_type := FETCHED_RECORD;
    end ;
-----

```

CURRENT STATE:

The application machine is in the origin connected state.

ENABLING CONDITION:

An A_FETCH_RECORD.request event occurs at the user interface and the phase is information retrieval.

ACTION:

A FETCH_RECORD APDU is created and transmitted to the target system by means of a P_DATA.request event initiated at the presentation layer interface.

RESULTANT STATE:

The application machine awaits a response to the fetch request.

```

-----
41. <FCH_WAIT> <--- <T_CON>
    [from P:P_DATA.indication]
      ((ptype(NULL) = FETCH_RECORD) and
       (phase = INFORMATION_RETRIEVAL))

    begin
      apdu_header := [from P:Object_list.header];
      [to U:A_FETCH_RECORD.indication
       (Fullness_code := pdux(FULLNESS_CODE),
        Record_key := pdux(RECORD_KEY),
        File_name := pdux(FILE_NAME))];
    end ;
-----

```

CURRENT STATE:

The application machine is in the target connected state.

ENABLING CONDITION:

A P_DATA.indication event occurs at the presentation layer interface. A FETCH_RECORD APDU has been received and the phase is information retrieval.

ACTION:

An A_FETCH_RECORD.indication event is initiated at the user interface. The key of the record is passed as a parameter to the end-user.

RESULTANT STATE:

The application machine awaits a response to the fetch record indication given to the end-user.

```

-----
42. <T_CON> <--- <FCH_WAIT>
    [from U:A_FETCH_RECORD.response]

    begin
      [to P:P_DATA.request
      (Operation := NULL,
      Object_list :=
        (HEADER, pduh(FETCHED_RECORD,
          [from U:Fetch_status])),
        (Z39_2, [from U:Data]))];
    end ;
-----

```

CURRENT STATE:

The application machine is awaiting a response to a fetch record indication given to the end-user.

ENABLING CONDITION:

An A_FETCH_RECORD.response event occurs at the user interface.

ACTION:

A FETCHED_RECORD APDU is created and transmitted to the originating system by means of a P_DATA.request event initiated at the presentation layer interface.

RESULTANT STATE:

The application machine returns to the target connected state to await another request from the originating system.


```

-----
43.  <O_CON> <--- <FCH_PEND>
      [ from U:P_DATA.indication ]
      (ptype(NULL) = FETCHED_RECORD)

begin
  apdu_header := [ from P:Object_list.header ];
  [ to U:A_FETCH_RECORD.confirm
    (Fetch_status := pdux(FETCH_STATUS),
      Data := [ from P:Object_list.data ] ) ];
end ;
-----

```

CURRENT STATE:

The application machine is awaiting the response to a fetch record request sent to the target system.

ENABLING CONDITION:

A P_DATA.indication event occurs at the presentation layer interface. A FETCHED_RECORD APDU has been received.

ACTION:

An A_FETCH_RECORD.confirm event is initiated at the end-user interface, supplying the fetched record.

RESULTANT STATE:

The application machine returns to the origin connected state to await another request from the end-user.

```

-----
44. <DIS_PEND> <--- <O_CON>
    [from U:A_DISCARD_RESULT.request]
      (phase = INFORMATION_RETRIEVAL)

begin
  [to P:P_DATA.request
    (Operation := NULL,
     Object_list :=
       (HEADER, pduh(DISCARD_RESULT_SET,
                     [from U:Result_set_ID])));
   exp_pdu_type := RESULT_SET_DISCARDED;
  end ;
-----

```

CURRENT STATE:

The application machine is in the origin connected state.

ENABLING CONDITION:

An A_DISCARD_RESULT.request event occurs at the user interface and the phase is as information retrieval.

ACTION:

A DISCARD_RESULT_SET APDU is created and transmitted to the target system by means of a P_DATA.request event initiated at the presentation layer interface.

RESULTANT STATE:

The application machine awaits a response to the discard request.

```

45. <DIS_WAIT> <--- <T_CON>
    [ from P:P_DATA.indication
      ((ptype(NULL) = DISCARD_RESULT_SET) and
        (phase = INFORMATION_RETRIEVAL));

    begin
      apdu_header := [ from P:Object_list.header];
      [ to U:A_DISCARD_RESULT.indication
        (Result_set-ID := pdux(RESULT_SET_ID))];
    end ;

```

CURRENT STATE:

The application machine is in the target connected state.

ENABLING CONDITION:

A P_DATA.indication event occurs at the presentation layer interface. A DISCARD_RESULT_SET APDU has been received and the phase is information retrieval.

ACTION:

An A_DISCARD_RESULT.indication event is initiated at the user interface. The result set ID is passed as a parameter to the end-user process.

RESULTANT STATE:

The application machine awaits a response to the discard indication given to the end-user.

```

46. <T_CON> <--- <DIS_WAIT>
    [ from U:A_DISCARD_RESULT.response ]

    begin
      [ to P:P_DATA.request
        (Operation := NULL,
          Object_list :=
            (HEADER, pduh(RESULT_SET_DISCARDED,
                          [ from U:Discard_status ])) ) ],
    end ;

```

CURRENT STATE:

The application machine is awaiting a response to a discard result set indication given to the end-user.

ENABLING CONDITION:

An A_DISCARD_RESULT_SET.response event occurs at the user interface.

ACTION:

A RESULT_SET_DISCARDED APDU is created and transmitted to the originating system by means of a P_DATA.request event initiated at the presentation layer interface.

RESULTANT STATE:

The application machine returns to the target connected state to await another request from the originating system.

```

47. <O_CON> <--- <DIS_PEND>
    [from U:P_DATA.indication]
      (ptype(NULL) = RESULT_SET_DISCARDED)

    begin
      apdu_header := [from P:Object_list.header];
      [ to U:A_DISCARD_RESULT.confirm
        (Discard_status := pdux(DISCARD_STATUS))];
    end ;

```

CURRENT STATE:

The application machine is awaiting the response to a discard result set request sent to the target system.

ENABLING CONDITION:

A P_DATA.indication event occurs at the presentation layer interface. A RESULT_SET_DISCARDED APDU has been received.

ACTION:

An A_DISCARD_RESULT.confirm event is initiated at the end-user interface.

RESULTANT STATE:

The application machine returns to the origin connected state to await another request from the originating system.

```

48. <ADD_PEND> <--- <O_CON>
    [from U:A_ADD_RECORD.request]
      (phase = RECORD_MAINTENANCE)

begin
  [to P:P_DATA.request
    (Operation := NULL,
     Object_list :=
       (HEADER, pduh(ADD_RECORD,
                     [from U:File_name],
                     [from U:Record_key])),
       (Z39_2, [from U:Data]))];
    exp_pdu_type := ADD_RECEIVED;
  end ;

```

CURRENT STATE:

The application machine is in the origin connected state.

ENABLING CONDITION:

An A_ADD_RECORD.request event occurs at the user interface and the phase is record maintenance.

ACTION:

An ADD_RECORD APDU is created and transmitted to the target system by means of a P_DATA.request event initiated at the presentation layer interface.

RESULTANT STATE:

The application machine awaits a response to the add record request.

```

49. <ADD_WAIT> <-- <T_CON>
    [from P:P_DATA.indication]
      ((ptype(NULL) = ADD_RECORD) and
       (phase = RECORD_MAINTENANCE))

    begin
      apdu_header := [from P:Object_list.header];
      [to U:A_ADD_RECORD.indication
        (File_name := pdux(FILE_NAME),
         Record_key := pdux(RECORD_KEY),
         Data := [from P:Object_list.data])];
    end ;

```

CURRENT STATE:

The application machine is in the target connected state.

ENABLING CONDITION:

A P_DATA.indication event occurs at the presentation layer interface. An ADD_RECORD APDU has been received and the phase is record maintenance.

ACTION:

An A_ADD_RECORD.indication event is initiated at the user interface. The record key, file name, and record to be added are passed as parameters to the end-user.

RESULTANT STATE:

The application machine awaits a response to the add record indication given to the end-user.

```

50. <T_CON> <--- <ADD_WAIT>
    [from U:A_ADD_RECORD.response]

    begin
      [to P:P_DATA.request
      (Operation := NULL,
      Object_list :=
        (HEADER, pduh(ADD_RECEIVED,
                      [from U:Add_status],
                      [from U:Transaction_ref_number]))]);
    end ;

```

CURRENT STATE:

The application machine is awaiting a response to an add record indication given to the end-user.

ENABLING CONDITION:

An A_ADD_RECORD.response event occurs at the user interface.

ACTION:

An ADD_RECEIVED APDU is created and transmitted to the originating system by means of a P_DATA.request event initiated at the presentation layer interface.

RESULTANT STATE:

The application machine returns to the target connected state awaiting another request from the originating system.

```

-----
51. <O_CON> <--- <ADD_PEND>
    [from U:P_DATA.indication]
      (ptype(NULL) = ADD_RECEIVED)

begin
  apdu_header := [from P:Object_list.header];
  [to U:A_ADD_RECORD.confirm
    (Add_status := pdux(ADD_STATUS),
     Transaction_ref_number :=
                                     pdux(TRANSACTION_REF_NUMBER))];
end ;
-----

```

CURRENT STATE:

The application machine is awaiting the response to an add record request sent to the target system.

ENABLING CONDITION:

A P_DATA.indication event occurs at the presentation layer interface. An ADD_RECEIVED APDU has been received.

ACTION:

An A_ADD_RECORD.confirm event is initiated at the end-user interface, supplying the add status and a transaction reference number.

RESULTANT STATE:

The application machine returns to the origin connected state to await another request from the end-user.

```

52. <REP_PEND> <--- <O_CON>
    [ from U:A_REPLACE_RECORD.request ]
      (phase = RECORD_MAINTENANCE)

begin
  [ to P:P_DATA.request
    (Operation := NULL,
      Object_list :=
        (HEADER, pduh(REPLACE_RECORD,
          [ from U:File_name],
          [ from U:Record_key],
          [ from U:Old_version_number])),
        (Z39_2, [ from U:Data]))];
    exp_pdu_type := REPLACE_RECEIVED;
  end ;

```

CURRENT STATE:

The application machine is in the origin connected state.

ENABLING CONDITION:

An A_REPLACE_RECORD.request event occurs at the user interface and the phase is record maintenance.

ACTION:

A REPLACE_RECORD APDU is created and transmitted to the target system by means of a P_DATA.request event at the presentation layer interface.

RESULTANT STATE:

The application machine awaits a response to the replace record request.

```

53. <REP_WAIT> <--- <T_CON>
    [from P:PDATA.indication]
      ((ptype(NULL) = REPLACE_RECORD) and
       (phase = RECORD_MAINTENANCE))

begin
  apdu_header := [from P:Object_list.header];
  [to U:A_REPLACE_RECORD.indication
    (File_name := pdux(FILE_NAME),
     Record_key := pdux(RECORD_KEY),
     Old_version_number := pdux(OLD_VERSION_NUMBER),
     Data := [from P:Object_list.data])];
end ;

```

CURRENT STATE:

The application machine is in the target connected state.

ENABLING CONDITION:

A P_DATA.indication event occurs at the presentation layer interface. A REPLACE_RECORD APDU has been received and the phase is record maintenance.

ACTION:

An A_REPLACE_RECORD.indication event is initiated at the user interface. Parameters specifying the record key, file name, and the replacement record are passed to the end-user process.

RESULTANT STATE:

The application machine awaits a response to the replace record indication given to the end-user.

```

54. <T_CON> <--- <REP_WAIT>
    [from U:A_REPLACE_RECORD.response]

    begin
      [ to P:P_DATA.request
        (Operation := NULL,
         Object_list :=
           (HEADER, pduh(REPLACE_RECEIVED,
                        [ from U:Replace_status ],
                        [ from U:Transaction_ref_number ]))));
    end ;

```

CURRENT STATE:

The application machine is awaiting a response to a replace record indication given to the end-user.

ENABLING CONDITION:

An A_REPLACE_RECORD.response event occurs at the user interface.

ACTION:

A REPLACE_RECEIVED APDU is created and transmitted to the originating system by means of a P_DATA.request event initiated at the presentation layer interface.

RESULTANT STATE:

The application machine returns to the target connected state to await another request from the originating system.


```

-----
55. <O_CON> <--- <REP_PEND>
    [ from U:P_DATA.indication ]
      (ptype(NULL) = REPLACE_RECEIVED)

    begin
      apdu_header := [ from P:Object_list.header ];
      [ to U:A_REPLACE_RECORD.confirm
        (Replace_status := pdux(REPLACE_STATUS),
          Transaction_ref_number :=
                                pdux(TRANSACTION_REF_NUMBER)) ];
    end ;
-----

```

CURRENT STATE:

The application machine is awaiting the response to a replace record request sent to the target system.

ENABLING CONDITION:

A P_DATA.indication event occurs at the presentation layer interface. A REPLACE_RECEIVED APDU has been received.

ACTION:

An A_REPLACE_RECORD.confirm event is initiated at the end-user interface.

RESULTANT STATE:

The application machine returns to the origin connected state to await another request from the end-user.

```

56. <DEL_PEND> <--- <O_CON>
    [from U:A_DELETE_RECORD.request]
      (phase = RECORD_MAINTENANCE)

begin
  [to P:P_DATA.request
    (Operation := NULL,
     Object_list :=
       (HEADER, pduh(DELETE_RECORD,
                     [from U:Record_key],
                     [from U:File_name],
                     [from U:Old_version_number]))));
    exp_pdu_type := DELETE_RECEIVED;
end ;

```

CURRENT STATE:

The application machine is in the origin connected state.

ENABLING CONDITION:

An A_DELETE_RECORD.request event occurs at the user interface and the phase is record maintenance.

ACTION:

A DELETE_RECORD APDU is created and transmitted to the target system by means of a P_DATA.request event initiated at the presentation layer interface.

RESULTANT STATE:

The application machine awaits a response to the delete record request.

```

57. <DEL_WAIT> <--- <T_CON>
    [from P:P_DATA.indication]
      ((ptype(NULL) = DELETE_RECORD) and
       (phase = RECORD_MAINTENANCE))

    begin
      apdu_header := [from U:Object_list.header];
      [to U:A_DELETE_RECORD.indication
        (Record_key := pdux(RECORD_KEY),
         File_name := pdux(FILE_NAME),
         Old_version_number := pdux(OLD_VERSION_NUMBER))];
    end ;

```

CURRENT STATE:

The application machine is in the target connected state.

ENABLING CONDITION:

A P_DATA.indication event occurs at the presentation layer interface. A DELETE_RECORD APDU has been received and the phase is record maintenance.

ACTION:

An A_DELETE_RECORD.indication event is initiated at the user interface. The record key, file name, and old version number are passed as parameters to the end-user.

RESULTANT STATE:

The application machine awaits a response to the delete record indication given to the end-user.


```

-----
58. <T_CON> <--- <DEL_WAIT>
    [from U:A_DELETE_RECORD.response]

    begin
      [to P:P_DATA.request
      (Operation := NULL,
      Object_list :=
        (HEADER, pduh(DELETE_RECEIVED,
                      [from U:Delete_status],
                      [from U:Transaction_ref_number])));
    end ;
-----

```

CURRENT STATE:

The application machine is awaiting a response to a delete record indication given to the end-user.

ENABLING CONDITION:

An A_DELETE_RECORD.response event occurs at the user interface.

ACTION:

A DELETE_RECEIVED APDU is created and transmitted to the originating system by means of a P_DATA.request event initiated at the presentation layer interface.

RESULTANT STATE:

The application machine returns to the target connected state to await another request from the originating system.

```

59. <O_CON> <--- <DEL_PEND>
    [ from U:P_DATA.indication ]
      (ptype(NULL) = DELETE_RECEIVED)

    begin
      apdu_header := [ from P:Object_list.header ];
      [ to U:A_DELETE_RECORD.confirm
        (Delete_status := pdux(DELETE_STATUS),
          Transaction_ref_number :=
            pdux(TRANSACTION_REF_NUMBER)) ];
    end ;

```

CURRENT STATE:

The application machine is awaiting the response to a delete record request sent to the target system.

ENABLING CONDITION:

A P_DATA.indication event occurs at the presentation layer interface. A DELETE_RECEIVED APDU has been received.

ACTION:

An A_DELETE_RECORD.confirm event is initiated at the end-user interface.

RESULTANT STATE:

The application machine returns to the origin connected state to await another request from the end-user.

```

-----
60. <ADV_PEND> <--- <O_CON>
    [from U:A_ADD_VALIDATED.request]
      (phase = RECORD_MAINT_REPORT)

begin
  [to P:P_DATA.request
   (Operation := NULL,
    Object_list :=
      (HEADER, pduh(ADD_VALIDATED,
                    [from U:Add_validated_report],
                    [from U:Transaction_ref_number],
                    [from U:Record_key],
                    [from U:Diagnostic])),
    (Z39_2, [from U:Data]))];
  exp_pdu_type := VALIDATION_ACK;
end ;
-----

```

CURRENT STATE:

The application machine is in the origin connected state.

ENABLING CONDITION:

An A_ADD_VALIDATED.request event occurs at the user interface and the phase is record maintenance reporting.

ACTION:

An ADD_VALIDATED APDU is created and transmitted to the target system by means of a P_DATA.request event initiated at the presentation layer interface.

RESULTANT STATE:

The application machine awaits a response to the add validated request.

```

61. <ADV_WAIT> <--- <T_CON>
    [from P:P_DATA.indication]
      ((ptype(NULL) = ADD_VALIDATED) and
       (phase = RECORD_MAINT_REPORT))

begin
  apdu_header := [from P:Object_list.header];
  [to U:A_ADD_VALIDATED.indication
   (Add_validated_report := pdux(ADD_VALIDATED_REPORT),
    Transaction_ref_number := pdux(TRANSACTION_REF_NUMBER),
    Record_key := pdux(RECORD_KEY),
    Diagnostic := pdux(DIAGNOSTIC),
    Data := [from P:Object_list.data])];
end ;

```

CURRENT STATE:

The application machine is in the target connected state.

ENABLING CONDITION:

A P_DATA.indication event occurs at the presentation layer interface. An ADD VALIDATED APDU has been received and the phase is record maintenance reporting.

ACTION:

An A_ADD_VALIDATED.indication event is initiated at the user interface. A report, reference number, and optional diagnostic information are passed as parameters to the end-user.

RESULTANT STATE:

The application machine awaits a response to the add validated indication given to the end-user.

```

62. <T_CON> <--- <ADV_WAIT>
    [ from U:A_ADD_VALIDATED.response ]

    begin
      [ to P:P_DATA.request
        (Operation := NULL,
          Object_list :=
            (HEADER, pduh(VALIDATION_ACK,
                          [ from U:Validation_status ])))]];
    end ;

```

CURRENT STATE:

The application machine is awaiting a response to an add validated indication given to the end-user.

ENABLING CONDITION:

An A_ADD_VALIDATED.response event occurs at the user interface.

ACTION:

An VALIDATION_ACK APDU is created and transmitted to the originating system by means of a P_DATA.request event initiated at the presentation layer interface.

RESULTANT STATE:

The application machine returns to the target connected state to await another request from the originating system.

```

63. <O_CON> <--- <ADV_PEND>
    [from U:P_DATA.indication]
      (ptype(NULL) = VALIDATION_ACK)

    begin
      apdu_header := [from P:Object_list.header];
      [to U:A_ADD_VALIDATION.confirm
        (Validation_status := pdux(VALIDATION_STATUS))];
    end ;

```

CURRENT STATE:

The application machine is awaiting the response to an add validated report sent to the target system.

ENABLING CONDITION:

A P_DATA.indication event occurs at the presentation layer interface. A VALIDATION_ACK APDU has been received.

ACTION:

An A_ADD_VALIDATED.confirm event is initiated at the end-user interface, supplying the validation status.

RESULTANT STATE:

The application machine returns to the origin connected state to await another request from the end-user.

```

64. <RPV_PEND> <--- <O_CON>
    [ from U:A_REPLACE_VALIDATED.request ]
      ( phase = RECORD_MAINT_REPORT )

begin
  [ to P:P_DATA.request
    ( Operation := NULL,
      Object_list :=
        ( HEADER, pduh( REPLACE_VALIDATED,
          [ from U:Replace_validated_report ],
          [ from U:Transaction_ref_number ],
          [ from U:New_version_number ],
          [ from U:Diagnostic ] ),
        ( Z39_2, [ from U:Data ] ) ) ];
    exp_pdu_type := VALIDATION_ACK;
  end ;

```

CURRENT STATE:

The application machine is in the origin connected state.

ENABLING CONDITION:

An A_REPLACE_VALIDATED.request event occurs at the user interface and the phase is record maintenance reporting.

ACTION:

A REPLACE_VALIDATED APDU is created and transmitted to the target system by means of a P_DATA.request event at the presentation layer interface.

RESULTANT STATE:

The application machine awaits a response to the replace validated report.

```

65. <RPV_WAIT> <--- <T_CON>
    [from P:PDATA.indication]
      ((ptype(NULL) = REPLACE_VALIDATED) and
       (phase = RECORD_MAINT_REPORT))

begin
  apdu_header := [from P:Object_list.header];
  [to U:A_REPLACE_VALIDATED.indication
    (Replace_validated_report :=
      pdux(REPLACE_VALIDATED_REPORT),
      Transaction_ref_number := pdux(TRANSACTION_REF_NUMBER),
      New_version_number := pdux(NEW_VERSION_NUMBER),
      Diagnostic := pdux(DIAGNOSTIC),
      Data := [from P:Object_list.data])];
end ;

```

CURRENT STATE:

The application machine is in the target connected state.

ENABLING CONDITION:

A P_DATA.indication event occurs at the presentation layer interface. A A_REPLACE_VALIDATED APDU has been received and the phase is record maintenance reporting.

ACTION:

An A_REPLACE_VALIDATED.indication event is initiated at the user interface. Parameters specifying the results of the validation are passed to the end-user.

RESULTANT STATE:

The application machine awaits a response to the replace validated indication given to the end-user.

```

-----
66. <T_CON> <--- <RPV_WAIT>
    [from U:A_REPLACE_VALIDATED.response]

    begin
      [to P:P_DATA.request
        {Operation := NULL,
         Object_list :=
           (HEADER, pduh(VALIDATION_ACK,
                        [from U:Validation_status]))});
    end ;
-----

```

CURRENT STATE:

The application machine is awaiting a response to a replace validated indication given to the end-user.

ENABLING CONDITION:

An A_REPLACE_VALIDATED.response event occurs at the user interface.

ACTION:

A VALIDATION_ACK APDU is created and transmitted to the originating system by means of a P_DATA.request event initiated at the presentation layer interface.

RESULTANT STATE:

The application machine returns to the target connected state awaiting another request from the originating system.

```

67. <O_CON> <--- <RPV_PEND>
    [from U:P_DATA.indication]
      (ptype(NULL) = VALIDATION_ACK)

    begin
      apdu_header := [from P:Object_list.header];
      [to U:A_REPLACE_VALIDATED.confirm
        (Validation_status := pdux(VALIDATION_STATUS))];
    end ;

```

CURRENT STATE:

The application machine is awaiting the response to a replace validated report sent to the target system.

ENABLING CONDITION:

A P_DATA.indication event occurs at the presentation layer interface. A VALIDATION_ACK APDU has been received.

ACTION:

An A_REPLACE_VALIDATION.confirm event is initiated at the end-user interface.

RESULTANT STATE:

The application machine returns to the origin connected state to await another request from the end-user.

```

-----
68. <DLV_PEND> <--- <O_CON>
    [ from U:A_DELETE_VALIDATED.request ]
      (phase = RECOVD_MAINT_REPORT)

begin
  [ to P:P_DATA.request
    (Operation := NULL,
      Object_list :=
        (HEADER, pduh(DELETE_VALIDATED,
                      [ from U:Delete_validated_report ],
                      [ from U:Transaction_ref_number ]))));
    exp_pdu_type := VALIDATION_ACK;
  end ;
-----

```

CURRENT STATE:

The application machine is in the origin connected state.

ENABLING CONDITION:

An A_DELETE_VALIDATED.request event occurs at the user interface and the phase is record maintenance reporting.

ACTION:

A DELETE_VALIDATED APDU is created and transmitted to the target system by means of a P_DATA.request event initiated at the presentation layer interface.

RESULTANT STATE:

The application machine awaits a response to the delete validated report.

```

-----
69. <DLV_WAIT> <--- <T_CON>
    [from P:P_DATA.indication]
      ((ptype(NULL) = DELETE_VALIDATED) and
       (phase = RECORD_MAINT_REPORT))

begin
  apdu_header := [from P:Object_list.header];
  [to U:A_DELETE_VALIDATED.indication
   (Delete_validated_report :=
    pdux(DELETE_VALIDATED_REPORT),
    Transaction_ref_number :=
    pdux(TRANSACTION_REF_NUMBER))];
end ;
-----

```

CURRENT STATE:

The application machine is in the target connected state.

ENABLING CONDITION:

A P_DATA.indication event occurs at the presentation layer interface. A DELETE_VALIDATED APDU has been received and the phase is record maintenance reporting.

ACTION:

An A_DELETE_VALIDATED.indication event is initiated at the user interface. The results of the delete validation are passed as parameters to the end-user.

RESULTANT STATE:

The application machine awaits a response to the delete validation indication given to the end-user.

```

70.  <T_CON> <--- <DLV_WAIT>
      [from U:A_DELETE_VALIDATED.response]

      begin
        [to P:P_DATA.request
          (Operation := NULL,
           Object_list :=
             (HEADER, pduh(VALIDATION_ACK,
                           [from U:Validation_status])));
        end ;

```

CURRENT STATE:

The application machine is awaiting a response to a delete validation indication given to the end-user.

ENABLING CONDITION:

An A_DELETE_VALIDATED.response event occurs at the user interface.

ACTION:

A VALIDATION_ACK APDU is created and transmitted to the originating system by means of a P_DATA.request event initiated at the presentation layer interface.

RESULTANT STATE:

The application machine returns to the target connected state to await another request from the originating system.

```

71. <O_CON> <--- <DLV_PEND>
    [from U:P_DATA.indication]
    (ptype(NULL) = VALIDATION_ACK)

    begin
        apdu_header := [from P:Object_list.header];
        [to U:A_DELETE_VALIDATED.confirm
        (Validation_status := pdux(VALIDATION_STATUS))];
    end ;

```

CURRENT STATE:

The application machine is awaiting the response to a delete validated report sent to the target system.

ENABLING CONDITION:

A P_DATA.indication event occurs at the presentation layer interface. A VALIDATION_ACK APDU has been received.

ACTION:

An A_DELETE_VALIDATED.confirm event is initiated at the end-user interface.

RESULTANT STATE:

The application machine returns to the origin connected state to await another request from the end-user.

```

72. <SML_PEND> <--- <O_CON>
    [from U:A_SEND_MAIL.request]
    (phase = MAIL_TRANSMISSION)

    begin
    [to P:P_DATA.request
    (Operation := NULL,
    Object_list :=
    (HEADER, pduh(SEND_MAIL,
    [from U:Mail_notification_mode],
    [from U:Recipient],
    [from U:Mail_reference_number])),
    (MAIL, [from U:Mail_text]))];
    exp_pdu_type := MAIL_RESPONSE;
    end ;

```

CURRENT STATE:

The application machine is in the origin connected state.

ENABLING CONDITION:

An A_SEND_MAIL.request event occurs at the user interface and the phase is mail transmission.

ACTION:

A SEND_MAIL APDU is created and transmitted to the target system by means of a P_DATA.request event initiated at the presentation layer interface.

RESULTANT STATE:

The application machine awaits a response to the mail transmission.

```

73. <SML_WAIT> <--- <T_CON>
    [ from P:P_DATA.indication]
      ((ptype(NULL) = SEND_MAIL) and
       (phase = MAIL_TRANSMISSION))

    begin
      apdu_header := [from P:Object_list.header];
      [ to U:A_SEND_MAIL.indication
        (Mail_notification_mode := pdux(MAIL_NOTIFICATION_MODE),
         Recipient := pdux(RECIPIENT),
         Mail_reference_number := pdux(MAIL_REFERENCE_NUMBER),
         Mail_text := [from P:Object_list.mail_text])];
    end ;

```

CURRENT STATE:

The application machine is in the target connected state.

ENABLING CONDITION:

A P_DATA.indication event occurs at the presentation layer interface. A SEND_MAIL APDU has been received and the phase is mail transmission.

ACTION:

An A_SEND_MAIL.indication event is initiated at the user interface. The mail notification mode, recipient, reference number, and text are passed as parameters to the end-user.

RESULTANT STATE:

The application machine awaits a response to the mail indication given to the end-user.

```

-----
74. <T_CON> <--- <SML_WAIT>
    [from U:A_SEND_MAIL.response]

    begin
      [to P:P_DATA.request
      (Operation := NULL,
      Object_list :=
        (HEADER, pduh(MAIL_RESPONSE,
                      [from U:Mail_status],
                      [from U:Mail_reference_number])));
    end ;
-----

```

CURRENT STATE:

The application machine is awaiting a response to a mail indication given to the end-user.

ENABLING CONDITION:

An A_SEND_MAIL.response event occurs at the user interface.

ACTION:

A MAIL_RESPONSE APDU is created and transmitted to the originating system by means of a P_DATA.request event initiated at the presentation layer interface.

RESULTANT STATE:

The application machine returns to the target connected state to await another request from the originating system.

```

-----
75. <O_CON> <--- <SML_PEND>
    [from U:P_DATA.indication]
      (ptype(NULL) = MAIL_RESPONSE)

    begin
      apdu_header := [from P:Object_list.header];
      [to U:A_SEND_MAIL.confirm
        (Mail_status := pdux(MAIL_STATUS),
         Mail_reference_number := pdux(MAIL_REFERENCE_NUMBER))];
    -----

```

CURRENT STATE:

The application machine is awaiting the response to a mail transmission sent to the target system.

ENABLING CONDITION:

A P_DATA.indication event occurs at the presentation layer interface. A MAIL_RESPONSE APDU has been received.

ACTION:

An A_SEND_MAIL.confirm event is initiated at the end-user interface.

RESULTANT STATE:

The application machine returns to the origin connected state to await another request from the end-user.

```

76. <MRD_PEND> <--- <O_CON>
    [from U:A_MAIL_READ.request]
      (phase = MAIL_READ_NOTIFICATION)

    begin
      [to P:P_DATA.request
        (Operation := NULL,
         Object_list :=
           (HEADER, pduh(MESSAGE_READ,
                         [from U:Mail_reference_number])));
        exp_pdu_type := MESSAGE_READ_RESPONSE;
      end ;

```

CURRENT STATE:

The application machine is in the origin connected state.

ENABLING CONDITION:

An A_MAIL_READ.request event occurs at the user interface and the phase is mail read notification.

ACTION:

A MESSAGE_READ APDU is created and transmitted to the target system by means of a P_DATA.request event initiated at the presentation layer interface.

RESULTANT STATE:

The application machine awaits a response to the mail read report.

```

-----
77. <MRD_WAIT> <--- <T_CON>
    [from P:P_DATA.indication]
      ((ptype(NULL) = MESSAGE_READ) and
       (phase = MAIL_READ_NOTIFICATION))

    begin
      apdu_header := [from P:Object_list.header];
      [to U:A_MAIL_READ.indication
        (Mail_reference_number := pdux(MAIL_REFERENCE_NUMBER))];
    end ;
-----

```

CURRENT STATE:

The application machine is in the target connected state.

ENABLING CONDITION:

A P_DATA.indication event occurs at the presentation layer interface. A MESSAGE_READ APDU has been received and the phase is mail read notification.

ACTION:

An A_MAIL_READ.indication event is initiated at the user interface. The mail reference number is passed as a parameter to the end-user.

RESULTANT STATE:

The application machine awaits a response to the mail read notification given to the end-user.

```

-----
78. <T_CON> <-- <MRD_WAIT>
    [from U:A_MAIL_READ.response]

    begin
      [to P:P_DATA.request
        (Operation := NULL,
         Object_list :=
           (HEADER, pduh(MESSAGE_READ_RESPONSE,
                        [from U:Mail_read_status]))];
    end ;
-----

```

CURRENT STATE:

The application machine is awaiting a response to a mail read notification given to the end-user.

ENABLING CONDITION:

An A_MAIL_READ.response event occurs at the user interface.

ACTION:

A MAIL_READ_RESPONSE APDU is created and transmitted to the originating system by means of a P_DATA.request event initiated at the presentation layer interface.

RESULTANT STATE:

The application machine returns to the target connected state to await another request from the originating system.

```

79. <O_CON> <--- <MRD_PEND>
    [from U:P_DATA.indication]
    (ptype(NULL) = MESSAGE_READ_RESPONSE)

    begin
      apdu_header := [from P:Object_list.header];
      [to U:A_MAIL_READ.confirm
       (Mail_read_status := pdux(MAIL_READ_STATUS))];

    end ;

```

CURRENT STATE:

The application machine is awaiting the response to a mail read notification sent to the target system.

ENABLING CONDITION:

A P_DATA.indication event occurs at the presentation layer interface. A MESSAGE_READ_RESPONSE APDU has been received.

ACTION:

An A_MAIL_READ.confirm event is initiated at the end-user interface.

RESULTANT STATE:

The application machine returns to the origin connected state to await another request from the end-user.

14. PROTOCOL DATA UNIT FORMATS

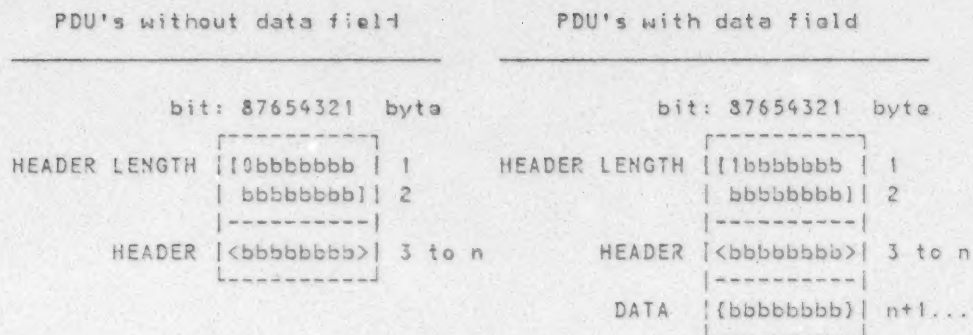
14.0 INTRODUCTION

14.0.1 NOTATION

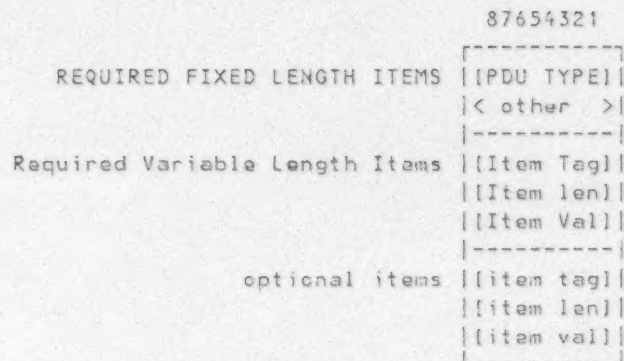
- [] enclose a fixed length data element;
a line beginning and/or ending with square brackets
contains one eight bit byte
- < > enclose a variable length data element containing an integral
number of eight bit bytes;
a line beginning or ending with angle brackets contains
an integral number of eight bit bytes
- { } enclose optional data elements
- 0 within brackets, indicates a bit set to zero
- 1 within brackets, indicates a bit set to one
- b within brackets, indicates a bit which may be set to either
zero or one
- _ within brackets, placed holder until code is assigned
- * indicates a repeatable item, or group of items
:
:
:
- + indicates a repeatable item, or group of items, nested under
| the occurrence of a higher level item
|

All lengths are unsigned binary numbers containing an integral
number of eight bit bytes. Lengths do not include themselves.

14.0.2 BASIC PDU FORMAT



14.0.3 BASIC HEADER FORMAT



14.1 PDU DATA ELEMENTS

Abort Reason - the reason the connection was disconnected;
one byte code: bit 8 = 0 for user initiated reasons
bit 8 = 1 for layer initiated reasons

00000000 USER REJECTED CONNECTION REQUEST
11111111 APPLICATION USER NOT AVAILABLE

Add Status - status of record add; one byte code:

00000000 SUCCESS
11111110 CANCELED
11111111 FAILURE

Add Validated Report - one byte code:

00000000 SUCCESS
11111111 FAILURE

Cancel Reason - the reason for the interrupt;
one byte code:

Class of Service - one byte code:

00000000 DEFAULT

Connection Parameter Item - one byte code: _____

length - binary, one byte integer
value - <code> <setting>

00000000 reserved
00000001 account parm
00000010 password parm
00000011 user name parm
00000100 max PDU len parm
00000101 timer-out parm
11111111 reserved
see section 9.2.2 for specific formats

Data Value - the value of the data portion of the PDU; character

Delete Status - status of record delete; one byte code:

00000000 SUCCESS
11111110 CANCELED
11111111 FAILURE

Discard Status - status of result set discard;
one byte code:

00000000 SUCCESS
10000001 UNKNOWN RESULT SET ID
11111110 CANCELED
11111111 FAILURE

Error Reason - The cause of the error; one byte code:

Fetch Status - status of the record fetch;
one byte code:

00000000 SUCCESS
10000001 NO SUCH RECORD KEY IN FILE
11111110 CANCELED
11111111 FAILURE

File Name Item : one byte code: _____
length - length of the file name; binary, one byte integer
value - value of the file name; character

Fullness Code - completeness level of the retrieved records;
one byte code (cf ranges)

Header Length - length of the pdu header, not including
itself; binary, two bytes. High order bit of the
first byte indicates the presence (1) or absence
(0) of data following the header.

Logical Operator Item : one byte code: _____
length - logical operator length; binary, one byte
integer
value - logical operator; character

AND
OR
NOT

Last PDU Type - pdu type of the previous pdu; one byte code:
see PDU TYPE for coding

Mail Notification Mode - "notification after read" requested,
or not requested; one byte code:

00000000 NOTIFICATION AFTER READ IS REQUESTED
11111111 NOTIFICATION AFTER READ IS NOT REQUESTED

Mail Reference Number Item - one byte code: _____
length - binary, one byte integer
value - character

Mail Status - status of a mail action; one byte code

00000000	SUCCESS
10000001	UNKNOWN RECIPIENT ID
10000010	RECIPIENT DOES NOT HAVE A MAILBOX
11111110	CANCELED
11111111	FAILURE

Mail Read Status - status of a mail read action; one byte code

00000000	SUCCESS
11111111	FAILURE

New Version Number Item: one byte code: _____
length - length of the value of the version identifier;
binary, one byte integer
value - value of the version identifier of the record;
character

Number of Records Item: one byte code: _____
length - length of the value of the number of records;
binary, one byte integer
value - number of records; binary integer

Old Version Number Item: one byte code: _____
length - length of the value of the version identifier;
binary, one byte integer
value - value of the version identifier of the record;
character

PDU TYPE - identifier of the PDU; one byte code:

00000000	ESTABLISH CONNECTION
00000001	ACCEPT CONNECTION
00000010	RELEASE CONNECTION
00000011	RELEASE ANSWER
00000100	SET
00000101	SHOW
00000110	SHOW RESPONSE
00000111	CANCEL
00001000	SET RESPONSE

00001011	ERROR
00001100	ADD RECORD
00001101	ADD RECEIVED
00001110	REPLACE RECORD
00001111	REPLACE RECEIVED
00010000	DELETE RECORD
00010001	DELETE RECEIVED
00010010	ADD VALIDATED
00010011	REPLACE VALIDATED
00010100	DELETE VALIDATED
00010101	VALIDATION ACK
00010110	SEARCH
00010111	SEARCH RESPONSE

00011000	PRESENT RECORDS
00011001	PRESENTED RECORDS
00011010	DISCARD RESULT SET
00011011	RESULT SET DISCARDED
00011100	FETCH RECORD
00011101	FETCHED RECORD
00011110	SEND MAIL
00011111	MAIL RESPONSE
00100000	MESSAGE READ
00100001	MESSAGE READ RESPONSE
00100010	SAVE RESULT SET
00100011	RESULT SET SAVED

Present Status - status of record presentation from the result set; one byte code:

00000000	SUCCESS
11111100	NO CURRENT RESULT
11111101	UNKNOWN RESULT SET NAME
11111110	CANCELED
11111111	FAILURE

Protocol Id - protocol identifier; one byte code:

00000001 LSP

Protocol Version - protocol version number; one byte code:

00000001 VERSION 1

Recipient Item : one byte code: _____

length - length of the mail recipient value; binary, one byte integer

value - mail recipient; character

Record Diagnostic Item - one byte code: _____

length - binary, one byte integer

value - character : index of error(s) in the record
<error code> <field tag> <occurrence number>

Record Key Item : one byte code: _____

length - length of the record key value; binary, one byte integer

value - value of the record key; character

Release Reason - The reason the connection was closed; one byte code:

00000000 NORMAL TERMINATION

Replace Status - status of the record replace; one byte code:

00000000	SUCCESS
11111110	CANCELED
11111111	FAILURE

Replace Validated Report - one byte code:

00000000 SUCCESS
11111111 FAILURE

Result Set Position Item - relative position within the result set; one byte code: _____
length - binary, one byte integer
value - binary integer

Result Set Id Item : one byte code: _____
length - length of the value of the result set identifier; binary, one byte integer
value - value of the result set identifier; character 'CURRENT'

Result Count - number of records in the result set; binary, two byte integer

RPN Query Structure : signals beginning of search structure
one byte code: _____
length : binary, one byte integer, always 0

Save Status - status of the save result set request;
one byte code:

00000000 SUCCESS
10000001 FAILURE, DUPLICATE NAME
11111110 CANCELED
11111111 FAILURE

Search Diagnostic Item -
one byte code: _____
length - binary, one byte integer
value - <error code> <item code> <item value>

Search Status - status of the search request; one byte code:

00000000 SUCCESS
11111101 NO SEARCH TO CONTINUE
11111110 CANCELED
11111111 FAILURE

Service Item - service option tag; one byte code: _____
e.g., files, indexes, subapplications, search facilities available
length - service option length; binary, one byte integer
value - <code> <value(s)>; character
sub-application ids
search terms
file ids
defaults
limits

State Code - id of the state of the protocol machine;
one byte code:

Statistics Item : one byte code: _____
length - statistic value length; binary, one byte integer
value - <code> <setting>; character

Subapplication Id - subapplication id; one byte code:

00000001 Record Maintenance
00000010 Record Maintenance Reporting
00000011 Search
00000100 Mail
00000101 Mail Report

Subapplication Parameter Item: one byte code: _____
length - binary, one byte integer
value - <code> <setting>
see section 9.2.2 for specific formats

00000000 reserved
00000001 default file name
00000010 small result limit parm
00000011 small result format
00000100 medium result limit parm
00000101 medium result format parm
00000110 large result limit parm
00000111 large result format parm
00001000 result set disposition parm
00001001 record maintenance validation mode
00001010 search time limit
11111111 reserved

Term Structure : signals beginning of search term structure;
one byte code: _____
length - binary, one byte integer; always zero

Term Attribute Item : one byte code: _____
length - search term attribute value length; one byte integer
value - <attribute type code> <attribute value>

Transaction Reference Number Item : one byte code: _____
length - length of the transaction reference number value;
binary, one byte integer
value - value of the transaction reference number; character

Validation Status - status of the record validation;
one byte code:

00000000 SUCCESS
11111111 FAILURE

Value Item - value of a search term; one byte code: _____
length - binary, one byte integer
value - character

14.2 PDU Formats

14.2.1 ESTABLISH CONNECTION

```
FIXED ITEMS=====
HEADER LENGTH          [ 0bbbbbbb
                        bbbbbbbb ]
PDU TYPE                [ 00000000 ]
Protocol Id             [ bbbbbbbb ]
Protocol Version        [ bbbbbbbb ]
Class Of Service        [ bbbbbbbb ]
Subapplication id       [ bbbbbbbb ]
optional items-----
*Connection Parameter item [ _____ ]
:           length        [ bbbbbbbb ]
:           value         < bbbbbbbb >
*Subapplication Parameter item [ _____ ]
:           length        [ bbbbbbbb ]
:           value         < bbbbbbbb >
=====
```

14.2.2 ACCEPT CONNECTION PDU

```
FIXED ITEMS=====
HEADER LENGTH          [ 0bbbbbbb
                        bbbbbbbb ]
PDU TYPE                [ 00000001 ]
Protocol Version        [ bbbbbbbb ]
Class of service        [ bbbbbbbb ]
optional items-----
*Connection Parameter item [ _____ ]
:           length        [ bbbbbbbb ]
:           value         < bbbbbbbb >
*Subapplication Parameter item [ _____ ]
:           length        [ bbbbbbbb ]
:           value         < bbbbbbbb >
=====
```

14.2.3 ABORT PDU

```
FIXED ITEMS=====
HEADER LENGTH          [ bbbbbbbb
                        bbbbbbbb ]
PDU TYPE                [ 11111111 ]
Abort Reason            [ bbbbbbbb ]
State Code              [ bbbbbbbb ]
data _____ (optional)
Data Value              (< bbbbbbbb >)
=====
```

14.2.4 RELEASE CONNECTION PDU

```
FIXED ITEMS=====
HEADER LENGTH          [ 0bbbbbbb
                        bbbbbbbb ]
PDU TYPE                [ 00000010 ]
Release Reason          [ bbbbbbbb ]
optional items-----
* Statistics Item      ([ _____ ]
:           length     [ bbbbbbbb ]
:           value      < bbbbbbbb >]
=====
```

14.2.5 RELEASE ANSWER PDU

```
FIXED ITEMS=====
HEADER LENGTH          [ 0bbbbbbb
                        bbbbbbbb ]
PDU TYPE                [ 00000011 ]
optional items-----
* Statistics Item      ([ _____ ]
:           length     [ bbbbbbbb ]
:           value      < bbbbbbbb >]
=====
```

14.2.6 SET PDU

```
FIXED ITEMS=====
HEADER LENGTH          [ 0bbbbbbb
                        bbbbbbbb ]
PDU TYPE                [ 00000100 ]
optional items-----
* Subapplication Parameter item [ _____ ]
:           length     [ bbbbbbbb ]
:           value      < bbbbbbbb >]
=====
```

14.2.7 SET RESPONSE PDU

```
FIXED ITEMS=====
HEADER LENGTH          [ 0bbbbbbb
                        00000000 ]
PDU TYPE                [ 00001000 ]
SET STATUS              [ bbbbbbbb ]
optional items-----
*Subapplication Parameter Item ([ bbbbbbbb ]
:           length     [ bbbbbbbb ]
:           value      < bbbbbbbb >]
=====
```

14.2.8 SHOW PDU

```

FIXED ITEMS=====
HEADER LENGTH          [ 0bbbbbbb
                        bbbbbbbb ]
PDU TYPE                [ 00000101 ]
optional items-----
*Service Item          ([ _____ ]
:      length          [ bbbbbb ]
:      value           < bbbbbbbb >
=====

```

14.2.9 SHOW RESPONSE PDU

```

FIXED ITEMS=====
HEADER LENGTH          [ 0bbbbbbb
                        bbbbbbbb ]
PDU TYPE                [ 00000110 ]
Subapplication Code     [ _____ ]
optional items-----
*Service item          ([ _____ ]
:      length          [ bbbbbbbb ]
:      value           < bbbbbbbb >
=====

```

14.2.10 CANCEL PDU

```

FIXED ITEMS=====
HEADER LENGTH          [ 00000000
                        00000010 ]
PDU TYPE                [ 00000111 ]
Cancel Reason          [ bbbbbbbb ]
=====

```


14.2.11 ERROR PDU

```

FIXED ITEMS=====
HEADER LENGTH      [ 1bbbbbbb
                   bbbbbbbb ]
PDU TYPE           [ 00001011 ]
Error Reason       [ bbbbbbbb ]
State Code         [ bbbbbbbb ]
data _____
Data Value         {< bbbbbbbb >} PDU in error
=====

```

14.2.12 ADD RECORD PDU

```

FIXED ITEMS=====
HEADER LENGTH      [ 1bbbbbbb
                   bbbbbbbb ]
PDU TYPE           [ 00001100 ]
optional items-----
File Name Item     ([ _____ ]
                    length [ bbbbbbbb ]
                    value  < bbbbbbbb >)
Record Key Item     ([ _____ ]
                    length [ bbbbbbbb ]
                    value  < bbbbbbbb >)
data _____
Data Value         < bbbbbbbb >
=====

```

14.2.13 ADD RECEIVED PDU

```

FIXED ITEMS=====
HEADER LENGTH          [ 0bbbbbbb
                        bbbbbbbb ]
PDU TYPE                [ 00001101 ]
Add Status              [ bbbbbbbb ]
optional items-----
Transaction Reference Num Item  ([ _____ ]
                                length [ bbbbbbbb ]
                                value  < bbbbbbbb > )
=====

```

14.2.14 REPLACE RECORD PDU

```

FIXED ITEMS=====
HEADER LENGTH          [ 1bbbbbbb
                        bbbbbbbb ]
PDU TYPE                [ 00001110 ]
optional items-----
File Name Item         ([ _____ ]
                        length [ bbbbbbbb ]
                        value  < bbbbbbbb > )
Record Key Item        ([ _____ ]
                        length [ bbbbbbbb ]
                        value  < bbbbbbbb > )
Old Version Number Item ([ _____ ]
                        length [ bbbbbbbb ]
                        value  < bbbbbbbb > )
data_____
Data Value              < bbbbbbbb >
=====

```

14.2.15 REPLACE RECEIVED PDU

```

FIXED ITEMS=====
HEADER LENGTH          [ 0bbbbbb ]
                        bbbbbbbb ]
PDU TYPE                [ 00001111 ]
Replace Status          [ bbbbbbbb ]
optional items-----
Transaction Reference Num Item  ([ _____ ]
                                length [ bbbbbbbb ]
                                value  < bbbbbbbb > }
=====

```

14.2.16 DELETE RECORD PDU

```

FIXED ITEMS=====
HEADER LENGTH          [ 1bbbbbbb ]
                        bbbbbbbb ]
PDU TYPE                [ 00010000 ]
Required Variable Length Items-----
Record Key Item        [ _____ ]
                        length [ bbbbbbbb ]
                        value  < bbbbbbbb >
optional items-----
File Name Item         ([ _____ ]
                        length [ bbbbbbbb ]
                        value  < bbbbbbbb > }
Old Version Number Item ([ _____ ]
                        length [ bbbbbbbb ]
                        value  < bbbbbbbb > }
=====

```

14.2.17 DELETE RECEIVED PDU

```

FIXED ITEMS=====
HEADER LENGTH          [ 0bbbbbbb ]
                        bbbbbbbb ]
PDU TYPE                [ 00010001 ]
Delete Status          [ bbbbbbbb ]
optional items-----
Transaction Reference Num Item  ([ _____ ]
                                length [ bbbbbbbb ]
                                value  < bbbbbbbb > }
=====

```


14.2.18 ADD VALIDATED PDU

```

FIXED ITEMS=====
HEADER LENGTH          [ bbbbbb ]
                        bbbbbb ]
PDU TYPE                [ 00010010 ]
Add Validated Report    [ bbbbbb ]
Required variable Length Items-----
Transaction Reference Num Item [ _____ ]
                        length [ bbbbbb ]
                        value  < bbbbbb >
Record Key Item         [ _____ ]
                        length [ bbbbbb ]
                        value  < bbbbbb >
optional items-----
Record Diagnostic Item   ([ _____ ]
                        length [ bbbbbb ]
                        value  < bbbbbb >
data_____
Data Value              {< bbbbbb >}
=====

```

14.2.19 REPLACE VALIDATED PDU

```

Fixed Items-----
HEADER LENGTH          [ bbbbbb ]
                        bbbbbb ]
PDU TYPE                [ 00010011 ]
Replace Validated Report [ bbbbbb ]
optional items-----
Transaction Reference Num Item ([ _____ ]
                        length [ bbbbbb ]
                        value  < bbbbbb >
New Version Number Item  ([ _____ ]
                        length [ bbbbbb ]
                        value  < bbbbbb >
Record Diagnostic Item   ([ _____ ]
                        length [ bbbbbb ]
                        value  < bbbbbb >
data_____
Data Value              {< bbbbbb >}
=====

```

14.2.20 DELETE VALIDATED PDU

```

FIXED ITEMS=====
HEADER LENGTH      [ bbbbbbbb
                   bbbbbbbb ]
PDU TYPE           [ 00010100 ]
DELETE VALIDATED REPORT [ bbbbbbbb ]
Optional item-----
Transaction Reference Num Item { [ _____ ]
                                length [ bbbbbbbb ]
                                value  < bbbbbbbb > }
data_____
Data Value          (< bbbbbbbb >)
=====

```

14.2.21 VALIDATION ACK PDU

```

FIXED ITEMS=====
HEADER LENGTH      [ 00000000
                   00000010 ]
PDU TYPE           [ 00010101 ]
Validation Status   [ bbbbbbbb ]
=====

```

14.2.22 SEARCH PDU

The Query Body described here is structured to support a query presented in Reverse Polish Notation. Other types of presentation could be developed and specified by a different Query Structure label. Entireties within the RPN Query Structure are to be ordered according to the conventions of standard post-fix notation.

```

FIXED ITEMS=====
HEADER LENGTH      [ 0bbbbbbb
                   bbbbbbbb ]
PDU TYPE           [ 00010110 ]
Qualification Indicator [ bbbbbbbb ]
RPN Query Structure [      ]
    Length         [ 00000000 ]
* Result Set Id Item ([      ]
:   length         [ bbbbbbbb ]
:   value          < bbbbbbbb >
* Term Structure    ( [      ]
:   Length         [ 00000000 ]
:   Value Item     [      ]
:   length         [ bbbbbbbb ]
:   value          < bbbbbbbb >
: + Term Attribute Item ([      ]
: |   length       [ bbbbbbbb ]
: |   value        [ bbbbbbbb ]})
* Logical Operator Item ([      ]
:   length         [ bbbbbbbb ]
:   value          [ bbbbbbbb ]}

optional items-----
File Name Item      ([      ]
    length          [ bbbbbbbb ]
    value           < bbbbbbbb >
=====

```


14.2.23 SEARCH RESPONSE PDU

```

FIXED ITEMS=====
HEADER LENGTH          [ bbbbbbbbb
                        bbbbbbbbb ]
PDU TYPE                [ 00010111 ]
Search Status          [ bbbbbbbbb ]
Result Count           [ bbbbbbbbb
                        bbbbbbbbb ]
Fullness Code          [ bbbbbbbbb ]
optional items-----
Result Set Id Item     ([ _____ ]
                        length      [ bbbbbbbb ]
                        value       < bbbbbbbb >)
Search Diagnostic Item ([ _____ ]
                        length      [ bbbbbbbbb ]
                        value       < bbbbbbbb >)
RPN Query Structure    ( [ _____ ]
                        length      [ 00000000 ]
* Result Set Id Item   ([ _____ ]
:      length          [ bbbbbbbbb ]
:      value           < bbbbbbbb >)
* Logical Operator Item ([ _____ ]
:      length          [ bbbbbbbbb ]
:      value           [ bbbbbbbb ]})
* Term Structure       ( [ _____ ]
:      length          [ 00000000 ]
:      Value Item.     [ _____ ]
:      length          [ bbbbbbbbb ]
:      value           < bbbbbbbb >
: + Term Attribute Item ([ _____ ]
: |      length        [ bbbbbbbbb ]
: |      value         [ bbbbbbbb ]})
* Logical Operator Item ([ _____ ]
:      length          [ bbbbbbbbb ]
:      value           [ bbbbbbbb ]})
Number of Records Item ([ _____ ]
                        length      [ bbbbbbbbb ]
                        value       < bbbbbbbb >

data_____
Data Value              (< bbbbbbbb >)
=====

```

14.2.24 PRESENT RECORDS PDU

```

FIXED ITEMS=====
HEADER LENGTH          [ 0bbbbbbb
                        bbbbbbbb ]
PDU TYPE                [ 00011000 ]
Fullness Code           [ bbbbbbbb ]
optional items-----
Result Set Position Item ([ _____ ]
                        length [ 00000010 ]
                        value  [ bbbbbbbb
                        bbbbbbbb ])
Number Of Records Item ([ _____ ]
                        length [ bbbbbbbb ]
                        value  < bbbbbbbb >)
Result Set Id Item      ([ _____ ]
                        length [ bbbbbbbb ]
                        value  < bbbbbbbb >)
=====

```

14.2.25 PRESENTED RECORDS PDU

```

Fixed Items -----
HEADER LENGTH          [ bbbbbbbb
                        bbbbbbbb ]
PDU TYPE                [ 00011001 ]
Present Status          [ bbbbbbbb ]
Required Variable Items-----
Result Set Position Item ([ _____ ]
                        length [ bbbbbbbb ]
                        value  < bbbbbbbb >)
Number Of Records Item ([ _____ ]
                        length [ bbbbbbbb ]
                        value  < bbbbbbbb >)
data _____
Data Value              {< bbbbbbbb >}
=====

```

14.2.26 DISCARD RESULT SET PDU

```
FIXED ITEMS=====
HEADER LENGTH      [ 0bbbbbbb
                   bbbbbbbb ]
PDU TYPE           [ 00011010 ]
Required Variable Length Items-----
Result Set Id Item [ _____ ]
    length         [ bbbbbbbb ]
    value          < bbbbbbbb >
=====
```

14.2.27 RESULT SET DISCARDED PDU

```
FIXED ITEMS=====
HEADER LENGTH      [ 00000000
                   00000010 ]
PDU TYPE           [ 00011011 ]
Discard Status     [ bbbbbbbb ]
=====
```


14.2.28 FETCH RECORD PDU

```

=====
FIXED ITEMS=====
HEADER LENGTH          [ 0bbbbbbb
                        bbbbbbbb ]
PDU TYPE               [ 00011100 ]
Fullness Code         [ bbbbbbbb ]
Required Variable Length Items-----
Record Key Item       [ _____ ]
                        length      [ bbbbbbbb ]
                        value       < bbbbbbbb >
optional items-----
File Name Item        ([ _____ ]
                        length      [ bbbbbbbb ]
                        value       < bbbbbbbb >)
=====

```

14.2.29 FETCHED RECORD PDU

```

=====
FIXED ITEMS=====
HEADER LENGTH          [ bbbbbbbb
                        bbbbbbbb ]
PDU TYPE               [ 00011101 ]
Fetch Status          [ bbbbbbbb ]
data _____
Data Value             (< bbbbbbbb >)
=====

```

14.2.30 SEND MAIL PDU

```

FIXED ITEMS=====
HEADER LENGTH          [ 1bbbbbbb
                        bbbbbbbb ]

PDU TYPE                [ 00011110 ]
Mail Notification Mode  [ bbbbbbbb ]
Required Variable Items-----
*Recipient Item        [ _____ ]
:      length          [ bbbbbbbb ]
:      value           < bbbbbbbb >
optional items-----
Mail Reference Number Item  ([ _____ ]
                           length [ bbbbbbbb ]
                           value  < bbbbbbbb > }

data _____
Data Value                < bbbbbbbb >
=====

```

14.2.31 MAIL RESPONSE PDU

```

FIXED ITEMS=====
HEADER LENGTH          [ 0bbbbbbb
                        bbbbbbbb ]

PDU TYPE                [ 00011111 ]
Mail Status             [ bbbbbb ]
optional items-----
Mail Reference Number Item  ([ _____ ]
                           length [ bbbbbbbb ]
                           value  < bbbbbbbb > }
=====

```

14.2.32 MESSAGE READ PDU

```

FIXED ITEMS=====
HEADER LENGTH      [ 0bbbbbbb
                   bbbbbbbb ]
PDU TYPE           [ 00100000 ]
Required Variable Length Items-----
Mail Reference Number Item [ _____ ]
                        length [ bbbbbbbb ]
                        value  < bbbbbbbb>
=====

```

14.2.33 MESSAGE READ RESPONSE PDU

```

FIXED ITEMS=====
HEADER LENGTH      [ 00000000
                   00000010 ]
PDU TYPE           [ 00100001 ]
Mail Read Status   [ bbbbbbbb ]
=====

```


REFERENCES

- [BBN-81/02] Bolt Beranek and Newman, Inc. An Automated Formal Specification Technique for Protocols. Cambridge, Mass., February, 1981. Working paper.
- [BCL-80/09] Hatelle Columbus Laboratories. Linking the Bibliographic Utilities: Benefits and Costs. Columbus, Ohio, September 15, 1980. Technical Report Submitted to the Council on Library Resources.
- [BIBNETT] Steering Committee of BIBNETT. Progress report from phase 3. "An application protocol for library and information systems". Oslo, Norway, March 13, 1981.
- [BIG-82/04] Computer Gateways, Inc. A File Transfer Service for the Bibliographic Interest Group. Ottawa, Canada, April 30, 1982. Draft.
- [CBOS-80-02] United States. Dept. of Commerce. National Bureau of Standards. Specification of a Draft Message Format Standard. October, 1980. Report no. ICST/CBOS-80-2.
- [CBOS-82-01] United States. Dept. of Commerce. National Bureau of Standards. Features of a Message Transfer Protocol. November, 1981. Report no. ICST/CBOS-82-1.
- [CBOS-82-03] United States. Dept. of Commerce. National Bureau of Standards. Service Specification of a Message Transfer Protocol. February, 1982. Report no. ICST/CBOS-82-3.
- [DP7498] International Organisation for Standardisation. Data Processing - Open Systems Interconnection - Basic Reference Model. December 3, 1980. Report no. ISO/DP7498, replaces ISO/TC97/SC16/N227.
- [FIPS-82/04] United States. Dept. of Commerce. National Bureau of Standards. Specification for Message Format for Computer Based Message Systems. April, 1982. Proposed Federal Information Processing Standard.
- [HLNP-80-01] United States. Dept. of Commerce. National Bureau of Standards. Features of the Transport and Session Protocols. March, 1980. Report no. ICST/HLNP-80-1.

- [HLNP-80-02] United States. Dept. of Commerce. National Bureau of Standards. Service Specification of Transport and Session Protocols. March, 1980. Report no. ICST/HLNP-80-2.
- [HLNP-80-03] United States. Dept. of Commerce. National Bureau of Standards. Formal Description Techniques for Network Protocols. June, 1980. Report no. ICST/HLNP-80-3.
- [HLNP-80-04] United States. Dept. of Commerce. National Bureau of Standards. Common Command Language Feature Analysis. June, 1980. Report no. ICST/HLNP-80-4.
- [HLNP-80-06] United States. Dept. of Commerce. National Bureau of Standards. Features of the File Transfer Protocol (FTP) and the Data Presentation Protocol (DPP). September, 1980. Report no. ICST/HLNP-80-6.
- [HLNP-80-07] United States. Dept. of Commerce. National Bureau of Standards. Formal Methods for Communication Protocol Specification and Verification. June, 1980. Report no. ICST/HLNP-80-7.
- [HLNP-80-08] United States. Dept. of Commerce. National Bureau of Standards. Features of Internetwork Protocol. July, 1980. Report no. ICST/HLNP-80-8.
- [HLNP-80-09] United States. Dept. of Commerce. National Bureau of Standards. Service Specification of the File Transfer Protocol (FTP) and the Data Presentation Protocol (DPP). October, 1980. Report no. ICST/HLNP-80-9.
- [HLNP-80-11] United States. Dept. of Commerce. National Bureau of Standards. Service Specification of an Internetwork Protocol. September, 1980. Report no. ICST/HLNP-80-11.
- [HLNP-80-12] United States. Dept. of Commerce. National Bureau of Standards. Features of Network Interprocess Communication Protocols. September, 1980. Report no. ICST/HLNP-80-12.
- [HLNP-80-15] United States. Dept. of Commerce. National Bureau of Standards. Service Specification of a Network Interprocess Communication Protocol. October, 1980. Report no. ICST/HLNP-80-15.

- [HLNP-81-02] United States. Dept. of Commerce. National Bureau of Standards. Specification of the Session Protocol. Revised March, 1981. Report no. ICST/HLNP-81-2.
- [HLNP-81-04] United States. Dept. of Commerce. National Bureau of Standards. Specification of the File Transfer Protocol. December, 1981. Report no. ICST/HLNP-81-4.
- [HLNP-81-05] United States. Dept. of Commerce. National Bureau of Standards. Specification of the Data Presentation Protocol. December, 1981. Report no. ICST/HLNP-81-5.
- [HLNP-81-07] United States. Dept. of Commerce. National Bureau of Standards. Feature Analysis of the Integrated Upper Levels Protocol. May, 1981. Report no. ICST/HLNP-81-7.
- [HLNP-81-08] United States. Dept. of Commerce. National Bureau of Standards. Service Features of the Integrated Upper Levels Protocol. May, 1981. Report no. ICST/HLNP-81-8.
- [HLNP-81-11] United States. Dept. of Commerce. National Bureau of Standards. Specification of the Transport Protocol, Volume 1: Overview and Services. September, 1981. Report no. ICST/HLNP-81-11.
- [HLNP-81-12] United States. Dept. of Commerce. National Bureau of Standards. Specification of the Transport Protocol, Volume 2: Basic Class Protocol. September, 1981. Report no. ICST/HLNP-81-12.
- [HLNP-81-13] United States. Dept. of Commerce. National Bureau of Standards. Specification of the Transport Protocol, Volume 3: Extended Class Protocol. September, 1981. Report no. ICST/HLNP-81-13.
- [HLNP-81-14] United States. Dept. of Commerce. National Bureau of Standards. Specification of the Transport Protocol, Volume 4: Network Interfaces. September, 1981. Report no. ICST/HLNP-81-14.
- [HLNP-81-15] United States. Dept. of Commerce. National Bureau of Standards. A Formal Specification Technique and Implementation Method for Protocols. July, 1981. Report no. ICST/HLNP-81-15.

- [HLNP-81-17] United States. Dept. of Commerce. National Bureau of Standards. The Impact of Satellite Transmission on High-level Computer Network Protocols. August, 1981. Report no. ICST/HLNP-81-17.
- [HLNP-81-18] United States. Dept. of Commerce. National Bureau of Standards. The Effects of Satellite Technology on the ISO Model of Open Systems Interconnection. August, 1981. Report no. ICST/HLNP-81-18.
- [HLNP-81-19] United States. Dept. of Commerce. National Bureau of Standards. Security in Higher Level Protocols: Approaches, Alternatives, and Recommendations. September, 1981. Report no. ICST/HLNP-81-19.
- [HLNP-81-20] United States. Dept. of Commerce. National Bureau of Standards. A Benchmark for Implementations of the NBS Transport Protocol. September, 1981. Report no. ICST/H[L]NP-81-20.
- [IEEE-80/04] The Institute of Electrical and Electronics Engineers, Inc. Communications Society. IEEE Transactions on Communications. "Special Issue on Computer Network Architectures and Protocols", April, 1980 (Volume COM-28, number 4). ISSN 0090-6778.
- [KUO-81] Kuo, Franklin F. Protocols and Techniques for Data Communication Networks. Englewood Cliffs, N.J., Prentice-Hall, 1981. ISBN 0-13-731729-8.
- [LASP-39] Washington Library Network, The Research Libraries Group, Inc., and Library of Congress. Linked Authorities Systems Project. Authorities Group. LSP External Design: Intersystem Component. February 26, 1982. Report no. LASP-39.
- [MFBD] United States. Library of Congress. Automated Systems Office. MARC Formats for Bibliographic Data. March 25, 1980. ISBN 0-8444-0333-4. Updated quarterly.
- [NTAG-77-80] NCLIS/NBS Task Force on Computer Network Protocol. Final Report and Recommendations. August, 1977. Second draft.
- [NTAG-78-21] Protocol Implementation Planners. Computer-to-Computer Communications Protocol Implementation Plan for the National Library and Information Service Network. New York, August 2, 1978. Final report to the Library of Congress Network Development Office.

- [NTAG-78-22] INSPEC, The Institution of Electrical Engineers.
EURONET Guideline: Standard Commands for Retrieval Systems. London, England, December, 1977. Final report on a study carried out for the Commission of the European Communities, DG XIII.
- [N0016] International Organisation for Standardisation.
Working Draft for Search and Operational Support Protocols (Command Language for Interrogation of Information Retrieval Systems). December, 1980. Report no. ISO/TC46/SC4/WG5/N16, replaces ISO/TC46/SC4/WG5/N13.
- [N0856] International Organisation for Standardisation. Draft Basic Connection-oriented Session Protocol Specification. January, 1982. Report no. ISO/TC97/SC16/N856.
- [N0898] International Organisation for Standardisation.
Guidelines for the Specification of Services for OSI. January, 1982. Report no. ISO/TC97/SC16/N898, replaces ISO/TC97/SC16/N577.
- [N0899] International Organisation for Standardisation.
Guidelines for the Specification of Protocols for OSI. January, 1982. Report no. ISO/TC97/SC16/N899, replaces ISO/TC97/SC16/N577.
- [N0918] International Organisation for Standardisation. Issues Pertinent to Application and Presentation Layers. March, 1982. Report no. ISO/TC97/SC16/N918.
- [N1161] International Organisation for Standardisation. Draft Connection Oriented Presentation Service Definition. June, 1982. Report no. ISO/TC97/SC16/N1161, replaces ISO/TC97/SC16/N919.
- [N1162] International Organisation for Standardisation.
Information Processing Systems - Open Systems Interconnection - Transport Service Definition. June, 1982. Report no. ISO/TC97/SC16/N1162.
- [N1169] International Organisation for Standardisation.
Information Processing Systems - Open Systems Interconnection - Transport Protocol Specification. June, 1982. Report no. ISO/TC97/SC16/N1169.

- [N1190] International Organisation for Standardisation.
Working Draft on File Transfer, Access and Management -
June 1982 - General Description. Report no.
ISO/TC97/SC16/N1190.
- [N1222] International Organisation for Standardisation.
Working Draft on File Transfer, Access and Management -
June 1982 - The Virtual Filestore. Report no.
ISO/TC97/SC16/N1222.
- [N1223] International Organisation for Standardisation.
Working Draft on File Transfer, Access and Management -
June 1982 - The File Service Definition. Report no.
ISO/TC97/SC16/N1223.
- [N1224] International Organisation for Standardisation.
Working Draft on File Transfer, Access and Management -
June 1982 - The File Protocol Specification. Report
no. ISO/TC97/SC16/N1224.
- [RFC-733] United States. Department of Defense. Advanced
Research Projects Agency. Committee on Computer-aided
Human Communication. Standard for the Format of ARPA
Network Text Messages. November 21, 1977. Request for
comments no. 733.
- [RR-80-43] International Institute for Applied Systems Analysis.
Remote File Access: A Data-access Protocol for
Computer Networks. Laxenburg, Austria, December, 1980.
Report no. RR-80-43. NTIS publication no. PB81-159840.
- [TAN-81] Tanenbaum, Andrew S. Computer Networks. Englewood
Cliffs, N.J., Prentice-Hall, 1981. ISBN 0-13-165183-8.
- [X3T5.5/82-121] American National Standards Institute.
Subcommittee X3T5.5. Rapporteur's Group on Upper Layer
Architecture. Common Application Layer Service Element
Definition. July 30, 1982. Report no. X3T5.5/82-121.
- [X3T5/82-209] Nielsen, F. Analysis of Standards Efforts -
Messaging and its Relationship to the OSI. Undated
letter to ANSI subcommittee X3T5. Report no.
X3T5/82-209.
- [Z39] Anonymous. Search and Retrieval in a
Computer-to-Computer Environment. Undated.
Unnumbered.